EMBEDDING METHODS FOR GENERATIVE MODELING AND VISUAL DATA ANALYSIS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Joshua Lamar Moore

February 2016

EMBEDDING METHODS FOR GENERATIVE MODELING AND VISUAL DATA ANALYSIS

Joshua Lamar Moore, Ph.D.

Cornell University 2016

In this work, we develop probabilistic embedding models for generative modeling and visual data analysis tasks. Embedding models are a class of models that assume the existence of a latent vector space representation for each of the modeled objects (where an object could be a word, a user, or a song or movie, depending on the task). In our models, we assume that this vector space defines a probability distribution over user choices or recommendations given a context of recent behavior. Furthermore, the scoring function which links the space to the goodness of fit between a choice and a context is chosen to focus on Euclidean distances in the embedding space. This allows us to build modular, interpretable models for a number of tasks. Although these models are generally applicable to many kinds of data, we mainly explore our applications of the models to tasks in the domain of Music Information Retrieval (MIR). First, we consider a playlist generation task which uses historical playlist logs to produce a model of good playlists, as well as a semantic genre space of the modeled songs. We then demonstrate the modularity of these models by adding side information and incorporating social tags to generalize to songs not seen in training, or out-of-vocabulary songs. We also demonstrate the power the interpretability of the semantic spaces of objects that result from our models. In the first case, we add temporal dynamics to our model in order to analyze trends and events in long-term music listening behavior logs. Finally, we apply

our model to a global data set of music plays in order to detect geographical, cultural, and linguistic patterns in music listening behavior in cities around the world.

BIOGRAPHICAL SKETCH

Joshua Lamar Moore was born in Griffin, Georgia and grew up in Jackson, Georgia, a small, rural town south of Atlanta. In 2010, he graduated with highest honors from the Georgia Institute of Technology with a Bachelor of Science in Computer Science, a Bachelor of Science in Applied Mathematics, and a minor in German. While studying at Georgia Tech, he had the privilege of conducting research in computer vision and robotics with Professor James M. Rehg. He also completed an exchange study at the Technical University of Munich in the 2007 to 2008 academic year, during which he worked as a research assistant in Professor Doctor Nassir Navab's Computer-Aided Medical Procedures and Augmented Reality (CAMPAR) group. After completing his undergraduate work, he worked for a summer with Professor Doctor Volker Tresp at Siemens in Munich. Since arriving at Cornell, he also completed summer internships at Google, Microsoft Research, and Facebook.

To my parents.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Thorsten Joachims, for his guidance, patience, and support over the course of my PhD. From the start, Thorsten did an excellent job of guiding a young researcher through forays into various topics of research. He was insightful and recognized my personal interest in music, introducing me to a related project that I was able to pursue with passion and enthusiasm. That project became this dissertation.

I would also like to thank the other members of my committee. Many thanks to Bobby Kleinberg, who also taught what was undoubtedly the most enriching course I completed at Cornell. Thanks to Wayne Harbert for graciously advising my linguistics minor and affording me an opportunity to learn about a topic I love outside my own field. I am forever indebted to Doug Turnbull for his great enthusiasm and invaluable perspective on the field of music information retrieval.

I would like to thank the professors who taught my linguistics courses: Sarah Murray, Miloje Despic, and John Whitman, for their patience in teaching a bit about their field to someone who knew almost nothing about it, as well as for the fascinating insights I gained into the nature and complexity of language.

Thanks to Jim Rehg, who advised me in research as an undergraduate and made it possible to pursue a doctoral degree to begin with. Thanks to Volker Tresp and Chris Burges, who both guided me on invaluable research internships and were excellent, supportive mentors. Thanks as well to Junfeng He, who mentored me during the final internship of my PhD and gave me an engaging summer of applying machine learning to real-world problems.

My time at Cornell would not have been the same without the great friendships I've formed here. I'm thankful for my great friends Alex Fix,

Deniz Altınbüken, Didier Chételat, Tom Magrino, Tobias Schnabel, Bilguun Erdenebat, Eoin O'Mahony, and Ploy Boonyaban, just to name a few. I owe a great deal of the friendships I formed at Cornell and many of my fondest memories to the Chapter House and the staff there – thank you for helping to provide a friendly community gathering place like no other. I was also helped along the way by some lasting friendships formed during summer internships, especially Aaron Gamble, C.J. Carey, John Gunderman, Theodora Hinkle, Aaron Stolarz, and Martin Kreichgauer. Several of my best friends from college made it easier to cope with grad school, especially Curtis Stephens, Victoria Au, Thomas Holmquist, Tejas Pargaonkar, and finally Daniel Mitchell, a true friend who will never be forgotten. My friends James Moore, Taylor Washington, and Jaime Ammons from home continued to provide indispensable support as well.

Finally, and most importantly, thanks to my family. My parents and my sister never ceased to provide me with encouragement, support, and love throughout my PhD, and I couldn't have gotten this far without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this dissertation, we aim to explore embedding spaces as meaningful, interpretable, and modular models for collections of log data for the purposes of generative modeling and data analysis. Namely, we wish to tackle the field of *Music Information Retrieval* (MIR) in order to solve problems related to user listening experiences and to analyze trends, events, and patterns in large records of music listening events. In order to address these two goals, we develop a number of embedding models, each suited to a particular task in this realm. These models combine a general formulation of generative modeling with the simplicity of assigning objects to a Euclidean space in a semantically meaningful way. This offers us two main advantages. First, the model yields positions for objects in a vector space wherein Euclidean distances carry a direct interpretation regarding the relationships among objects. This makes the resulting models easily interpretable and enables us to glean patterns of similarity from the data easily, even in a visual fashion, despite starting only with log data relating to co-occurrences, music listening events, and so on. Second, the formulation of this probabilistic model is highly general and modular in the sense that it is straightforward to adapt to new sources of side information or new modeling problems altogether. We will demonstrate the first of these through a series of novel analyses of patterns and trends in music data sets, and the latter through

the application of our methods to several different modeling problems.

1.1 Playlist Generation

As an initial foray into the application of embeddings to MIR, we consider the task of playlist generation, a specific instance of the more general problem of sequence modeling. In this scenario, we wish to develop a model that is capable of generating "good" sequences of songs for a listener. For this task, we create a model that assigns songs to a Euclidean space such that songs that are close together in the space also make sense together in a playlist. We demonstrate that our embedding model makes sense for this task by a comparison against standard baselines for sequence modeling tasks. In addition, visual inspection of the resulting embedding spaces demonstrates the potential of these methods for creating interpretable models.

1.2 Incorporating Side Information

A general problem in sequence modeling tasks (such as our playlist modeling problem) is the *out-of-vocabulary* (or OOV) problem. This refers to a situation where a model must handle objects (in our case, songs) at test time that were never observed in training time. We develop a model that incorporates social tags, explicit descriptors that provide a useful source of side information about songs, in order to generalize to new songs. In addition to solving the OOV problem for playlist modeling, our method is more generally applicable and could be used to alleviate the problem for other sequence modeling tasks as well. Furthermore, our model embeds the social tags in a joint space with the

songs. This allows us to use the resulting space to measure similarity between a song and an explicit description, as well as for query-by-tag retrieval tasks. The presence of explicit descriptions in the embedding space also further improves the interpretability of our model.

1.3 Temporal Data Analysis

Inspired by the ease of interpretability and visualization of our playlist generation models, we seek to develop models with the specific intent of analyzing log data. In particular, we develop an embedding model with temporal dynamics that can be used to analyze logs of users' music listening behavior over time. In these models, we consider the relationship between users and songs by embedding both in a common space. Temporal dynamics are added by dividing the data into discrete time steps and modeling each time step separately, but not indpeendently, from the others. Namely, objects are allowed to change position smoothly from one time step to the next. We apply this model to a collection of user listening histories spanning eight years. As a result, we are able to detect important events and changes in listening behavior in the data in a visual and spatial fashion.

1.4 Geographical Analysis

We further demonstrate the utility of our models for data analysis tasks by considering a geographical data analysis task. In this case, we consider a data set of music listening events paired with geographical information in the form of

GPS coordinates and the corresponding city, state, and country in which the event occurred. We then jointly embed cities and musical artists in a space such that cities are close to artists which are likely to be played in that city. Cities that are near each other in this space have similar probability distributions over artists, or in other words, similar taste in music. Using this fact, we can analyze patterns of similarity among cities around the world. This enables us to discover geographical, cultural, and linguistic patterns in the listening preferences of various cities.

CHAPTER 2

EMBEDDING MODELS

The most general form of the model class we consider uses a so-called *scoring function* $s(c, o)$ to relate a *context* c to a *target object* o. The meaning of the context and target object vary from one problem to the next. In general, the context defines a current state of the system, and the target object is a candidate for the next object to be chosen by the system. For example, in the case of playlist generation, the context consists of the previously played song, and the target object could be any known song which is being considered for the next track play. In the case of a recommender system, the context may be the user for whom we want to recommend a song, and the target object could again be any song in the data set. When the scoring function produces a large value for a given context and target object, this signifies that the target object in question is a good fit for the context. In our running examples, this would mean that the song is a good transition from the previously played song, or the song would be a good recommendation for the user. The precise definition of the scoring function varies from one model to the next and will be discussed in further detail later, but generally it relates some learned vector space representation of the objects in the context to a similar learned representation of the target object.

2.1 Generative Formulation

The models we use are generative. In particular, we model a set of conditional distributions $P(o|c)$ for each possible context. These give us the probability of the model selecting, e.g., a certain target song given the previously played song.

The scoring functions we use potentially produce any real number, and the values produced for various target objects could be independent of one another. Therefore, in order to produce probability distributions, we map the scoring function to a non-negative number using an exponential, and we normalize the probabilities for a given context to sum to one. This produces a probability of the form:

$$P(o|c) = \frac{\exp(s(c,o))}{Z(c)} \tag{2.1}$$

where $Z(c)$, known as the *partition function*, involves a sum over all possible target objects o in the *vocabulary* V:

$$Z(c) = \sum_{v \in V} \exp(s(c,v)) \tag{2.2}$$

Note that the partition function involves a sum over $|V|$ objects, and that this sum varies from one context to the next. This formulation is used in a number of works [14, 1, 25], but differs from the work of Mikolov et al [29, 28] where the probabilistic nature of the model is sacrificed in the name of efficiency. Our models generally involve problems where the number of contexts can be $O(|V|)$ or larger, which means that computing every partition function exactly could cost $O(|V|^2)$ or more. This poses a challenge in terms of efficient training and scalability, which will be discussed in Section 2.3. Since the model is generative, we use the average log-likelihood on a data set as a measure of model fit. Namely, given a data set D consisting of a number N of observed context-object pairs (c, o), the log-likelihood is given as:

$$\begin{aligned} LL(D) &= \frac{1}{N} \log\Big(\prod_{(c,o)\in D} P(o|c)\Big) \\ &= \frac{1}{N} \sum_{(c,o)\in D} \log\Big(\frac{\exp(s(c,o))}{Z(c)}\Big) \\ &= \frac{1}{N} \sum_{(c,o)\in D} s(c,o) - \log(Z(c)) \end{aligned} \tag{2.3}$$

If we further define C to be the set of all observed contexts and abuse notation to overload C to be the observed distribution over contexts and D to be the observed distribution over context-target pairs, we can also express this log-likelihood as an expectation over these distributions:

$$LL(D) = \mathop{\mathbb{E}}_{(c,o)\sim D}[s(c,o)] - \mathop{\mathbb{E}}_{c\sim C}[\log(Z(c))] \tag{2.4}$$

Training involves fitting the model parameters to the data by maximizing this log-likelihood function. Using the expectation form of the objective function, we train using stochastic gradient descent with a sampling approach based on context-object pairs (c, o). The full details of our training approach are given in Section 2.3.

2.2 Scoring Functions and Embeddings

The basis of our models is the concept of an *embedding function*. Such a function takes an arbitrary discrete object as an input, and returns a representation for that object in $\mathbb{R}^d$. This is a contrast to works in, e.g., Neural Probabilistic Language Models[5, 3, 4], where the scoring function takes a complex form such as a neural network, but similar to some works where a simple Euclidean distance or inner product score is used[40, 41, 29, 28]. The dimension d of the space is a parameter of the model which should be tuned for the task, and the resulting mappings – the individual coordinates of each vector for each object – are the parameters to be learned in training.

The representations of the objects are then used as inputs for the scoring

function, and the model is fit to the data using the objective explained in the previous section. Throughout our work, we choose scoring functions which are mainly Euclidean in nature. That is, using the learned vector space positions for the objects in the data set, we generally say that a particular target object is well-suited for a context if its representation is close in Euclidean distance to the representations of the objects in the context. Therefore, if we consider c to be an ordered tuple of objects $(c_1, c_2, ..., c_n)$, $I(c)$ to be the set of indices for this tuple, and let $\Delta(x, y) = \|x - y\|_2^2$ for vectors x and y, we obtain the following generic formulation for a scoring function:

$$s(c, o) = \sum_{i \in I(c)} -\Delta(X_i(c_i), Y_i(o)) + b_o \tag{2.5}$$

Here, each X_i and Y_i denotes an embedding function. This formulation includes a bias term b_o for the target object, which can take any real value and allows the model to place more probability mass on very popular objects without distorting the space to place all objects close to popular ones. The following should be noted about this formulation:

1. There is a distinct embedding function X_i for each object c_i in the context. In the case that each item in the tuple is a different type of object (for example, c_1 is a user but c_2 is a song), this is strictly necessary. However, in the case that all items in the context are the same type of object (such as generating the next song given the last three), it is still (optionally) possible to define a distinct embedding function for the same object given its position in the tuple. For example, song 17 might take a different position when it appears in the first component of the context than it does when it appears in the second.
2. There is a distinct embedding function Y_i for the target object depending

on which item of context is being considered. In other words, the target object can optionally take a different representation depending on the element of the context to which it is being related.

3. This formulation covers the models used in this work, but other variations are possible. For example, one could define the scoring function $s(c, o) = -\Delta(A(c, X), Y(o))$ where X and Y are embedding functions and A is an averaging operator defined as $A(c, X) = \frac{1}{|I(c)|} \sum_{i \in I(c)} X(c_i)$.

Euclidean formulations like these allow us to easily make semantic sense of the spaces that result from the data. As long as we design sensible scoring functions, then we know that songs that appear near each other are similar in the setting of a playlist, users that cluster together have similar taste in the setting of a recommendation system, and so on. We gain the particular advantage of directly visualizable models in the case where $d = 2$. In this scenario, it is actually possible to plot and inspect the resulting model. We will show later that this is a useful tool for gaining insight into large sets of log data.

Furthermore, the models we have described is very general – the setting of relating objects via arbitrary Euclidean "links" described in Equation 2.5 allows this type of model to apply to numerous settings with very different needs. The general formulation of the probabilistic model which takes the scoring function as a component is also highly modular, and we will see throughout this work how it can be easily adapted to the task at hand.

2.3 Scalable Training

In this chapter, we describe the method we use for training our models throughout the work. As mentioned previously, efficient training of these models is non-trivial due to the presence of the normalizing partition function, which is required for the model to yield probability distributions. Alternative methods for overcoming this challenge exist, such as those found in the works of Mnih et al [31, 30] and the works of Bengio et al [5, 4], detailed in Section 2.4. As we will show in Chapter 3, our training method is competitive with the state of the art in obtained objective for our data while also offering significantly faster convergence. In addition to this, a further advantage of our training method is its profound simplicity. We need only a single application of Jensen's inequality to justify it, the resulting algorithm can be written in just a few lines, and the method requires no hyper-parameters beyond those necessary for performing stochastic gradient descent.

2.3.1 Objective Bound and Partition Function Estimation

Recall from earlier in the chapter that the log-likelihood objective can be phrased in terms of expectations:

$$LL(D) = \mathop{\mathbb{E}}_{(c,o)\sim D}[s(c,o)] - \mathop{\mathbb{E}}_{c\sim C}[\log(Z(c))] \tag{2.6}$$

In this case, D is overloaded to signify the empirical distribution over context-object pairs found in the data, and C is similarly the empirical distribution over the contexts found in the data. We will refer to the expectation on the left as the

attraction term and the expectation on the right as the *repulsion term*. The complication in training arises from the computation of $Z(c)$, which must be computed independently for each context c. To sidestep this, note that we can lower-bound the log-likelihood using Jensen's inequality, moving the expectation inside the log:

$$LL(D) \geq \mathop{\mathbb{E}}_{(c,o)\sim D}[s(c,o)] - \log(\mathop{\mathbb{E}}_{c\sim C}[Z(c)]) \tag{2.7}$$

Note that this bound is normally fairly tight – the log function approaches a linear function for large inputs, and most embedding models have large partition functions. Let $\hat{Z} = \mathop{\mathbb{E}}_{c\sim C}[Z(c)]$. Now if U is the uniform distribution over V, consider that:

$$\begin{aligned} \hat{Z} &= \mathop{\mathbb{E}}_{c\sim C}[Z(c)] \\ &= |V| \mathop{\mathbb{E}}_{c\sim C}\left[\sum_{v\in V} \frac{\exp(s(c,v))}{|V|}\right] \\ &= |V| \mathop{\mathbb{E}}_{c\sim C}\left[\mathop{\mathbb{E}}_{v\sim U}[\exp(s(c,v))]\right] \end{aligned} \tag{2.8}$$

This means that it is possible to estimate $\hat{Z}$ by repeatedly sampling a context c from C, a target object v from U, computing $s(c,v)$, and appropriately summing and scaling the computed terms. This also yields an unbiased estimate. Now consider tracking $\hat{Z}$ over the course of the optimization. That is, to train the model, we repeatedly sample $(c,o) \sim D$ and $v \sim U$, update the estimate $\hat{Z}$, and update the parameters of the model corresponding to c, o, and v. In this scheme, we potentially incorporate "stale" samples into the estimate $\hat{Z}$, which means that we lose the advantage of this estimate being unbiased. However, heuristically, the actual value of $\hat{Z}$ should not change very rapidly between samples, since there are many parameters of which we update only a few and since

we only take a small gradient step in each parameter. In practice, we use a circular buffer to estimate $\hat{Z}$. In other words, we create an array B of size $|B|$, fill it initially, and store the sum in a variable t. Then, on the ith sample of the optimization we obtain a score s, add $\exp(s) - B[i \mod |B|]$ to t, and set $B[i \mod |B|] = \exp(s)$. In this way we get an estimate of $\hat{Z}$ by computing $\frac{|V|t}{|B|}$. The size of the buffer is a free parameter, but in general setting $|B| = |V|$ seems effective, so we will assume this buffer size from here on, meaning that t directly yields our estimate of $\hat{Z}$.

2.3.2 Gradient Updates

Now we will derive the general form of the gradient updates. First consider the partial derivative of the log-likelihood bound $LL_b(D)$ in Equation 2.7 with respect to any parameter ϕ:

$$\begin{aligned}
\frac{\partial LL_b(D)}{\partial \phi} &= \mathop{\mathbb{E}}_{(c,o)\sim D}\left[\frac{\partial s(c,o)}{\partial \phi}\right] - \mathop{\mathbb{E}}_{c\sim C}\left[\frac{1}{\hat{Z}}\frac{\partial Z(c)}{\partial \phi}\right] \\
&= \mathop{\mathbb{E}}_{(c,o)\sim D}\left[\frac{\partial s(c,o)}{\partial \phi}\right] - \mathop{\mathbb{E}}_{c\sim C}\left[\sum_{v\in V}\frac{1}{\hat{Z}}\exp(s(c,v))\frac{\partial s(c,v)}{\partial \phi}\right] \\
&= \mathop{\mathbb{E}}_{(c,o)\sim D}\left[\frac{\partial s(c,o)}{\partial \phi}\right] - \mathop{\mathbb{E}}_{c\sim C}\left[\mathop{\mathbb{E}}_{v\sim U}\left[\frac{|V|}{\hat{Z}}\exp(s(c,v))\frac{\partial s(c,v)}{\partial \phi}\right]\right] \\
&= \mathop{\mathbb{E}}_{(c,o)\sim D}\left[\frac{\partial s(c,o)}{\partial \phi}\right] - \mathop{\mathbb{E}}_{c\sim C}\left[\mathop{\mathbb{E}}_{v\sim U}\left[|V|\hat{P}(v|c)\frac{\partial s(c,v)}{\partial \phi}\right]\right]
\end{aligned} \tag{2.9}$$

where $\hat{P}(v|c)$ denotes the estimate of the model probability $P(v|c)$ given the current partition function estimate $\hat{Z}$. Recall the generic formulation of the scoring

function:

$$s(c, o) = \sum_{i \in I(c)} -\Delta(X_i(c_i), Y_i(o)) + b_o \tag{2.10}$$

Given a context c and an object o, the gradients with respect to the parameter vectors $X_i(c_i)$ and $Y_i(o)$ from the ith term of this sum, as well as the partial derivative with respect to the bias term b_o, are:

$$\begin{aligned} \frac{\partial s(c, o)}{\partial X_i(c_i)} &= -2(X_i(c_i) - Y_i(o)) \\ \frac{\partial s(c, o)}{\partial Y_i(o)} &= 2(X_i(c_i) - Y_i(o)) \\ \frac{\partial s(c, o)}{\partial b_o} &= 1 \end{aligned} \tag{2.11}$$

2.3.3 Stochastic Gradient Descent Training Algorithm

With these equations in hand, we can now define our sampling-based optimization algorithm. This algorithm takes a learning rate η as a parameter for stochastic gradient descent and is given in Algorithm 1.

When sampling a context-object pair $(c, o) \sim D$, the context c is also distributed according to the empirical context distribution C. Therefore, for any parameter ϕ, the update in expectation under this scheme is equal to the update in Equation 2.9.

The algorithm continues sampling and updating until some definition of convergence is met. One method of determining convergence is to track the estimate of the log-likelihood bound $LL_b(D)$ and terminate when this stops improving. This can be performed by additionally computing $s(c, o)$ at each step, keeping track of an estimate of $\mathop{\mathbb{E}}_{(c,o)\sim D}[s(c, o)]$, and periodically calculating

Algorithm 1: Training algorithm for embeddings

Initialize all $X_i(\cdot), Y_i(\cdot)$ to random points in unit ball in $\mathbb{R}^d$
Set all $b_o = 0$
Set B to an array of size $|V|$
Set $\hat{Z} = 0$
for j in 1 to $|V|$ **do**
    Sample $c \sim C$, $v \sim U$
    Set $B[j] = \exp(s(c, v))$
    Set $\hat{Z} = \hat{Z} + B[j]$
end for
Set $j = 0$
while not converged **do**
    Set $j = (j \mod |V|) + 1$
    Sample $(c, o) \sim D$, $v \sim U$
    Compute $s(c, v)$
    Set $\hat{Z} = \hat{Z} + \exp(s(c, v)) - B[j]$
    Set $B[j] = \exp(s(c, v))$
    Set $\hat{P}(v|c) = \exp(s(c, v)) \,/\, \hat{Z}$
    for i in $I(c)$ **do**
        Set $X_i(c_i) = X_i(c_i) - 2\eta(X_i(c_i) - Y_i(o))$ ▷ Attraction term updates
        Set $Y_i(o) = Y_i(o) + 2\eta(X_i(c_i) - Y_i(o))$
        Set $b_o = b_o + \eta$
        Set $X_i(c_i) = X_i(c_i) + 2\eta|V|\hat{P}(v|c)(X_i(c_i) - Y_i(v))$ ▷ Repulsion term updates
        Set $Y_i(v) = Y_i(v) - 2\eta|V|\hat{P}(v|c)(X_i(c_i) - Y_i(v))$
        Set $b_v = b_v - \eta|V|\hat{P}(v|c)$
    end for
end while

$\mathop{\mathbb{E}}_{(c,o)\sim D}[s(c,o)] - \log(\hat{Z})$. Alternatively, the objective can be calculated on a held-out validation set, and the optimization can be terminated when this objective stops improving. In this case, either the exact log-likelihood $LL(D)$ can be calculated, or for the sake of efficiency the bound $LL_b(D)$ can be calculated.

In practice, our learning rate η is not fixed. We use an adaptive method known as AdaGrad developed by Duchi et al [12] which uses knowledge of the geometry of the data gained from previous updates in order to more appropriately set the learning rate for each individual parameter. Specifically, we keep track of a buffer variable A_ϕ for each parameter ϕ. The buffer begins at zero, and each time we wish to update ϕ by a value u, we increase A_ϕ by u^2. Then, the update $\frac{\eta}{\sqrt{A_\phi}}u$ is added to ϕ. This essentially means that parameters which have already received large updates are changed with a small step size, while parameters that have only changed slowly or rarely receive larger updates. This is particularly advantageous for our sampling scheme – late in the optimization, the parameters corresponding to rarely occurring objects receive larger updates to compensate for the low frequency with which they are sampled.

The training procedure outlined in this chapter is generally applicable to the models in the remainder of the work. The models we define in Chapters 4 and 5 require only small modifications, which will be outlined in those parts of the work. In the next chapter, we will demonstrate the application of our models to the problem of playlist generation.

2.4 Related Work

Fundamentally, our models rely on the use of low-dimensional representations of objects and generative modeling. Many previous works consider models involving one or both of these concepts. Our contributions can be distinguished from these by considering our novel adaptations and applications for the realm of Music Information Retrieval as well as our use of these models for analysis of music listening histories and preferences.

2.4.1 Low-Dimensional Representations

One key aspect of our work is the concept of learning a low-dimensional representation of the objects in the data set. This is accomplished by the learned embedding function – we treat each coordinate of the representation of each object as a parameter in the model, and we perform stochastic gradient descent on these parameters to maximize the likelihood of the model generating the training data.

Multi-Dimensional Scaling (MDS) [9] is used to learn a low-dimensional representation of objects such that the pairwise distances between any two points in the new space directly approximates the dissimilarity indicated between the two represented objects. This method learns a space from the dissimilarity itself, as opposed to performing dimensionality reduction as in methods such as PCA [19], where an existing vector representation of the data is required. In this sense, MDS is similar to our embedding methodology. However, our methods offer a generative model based on the similarity of objects in the resulting

space. In addition, embedding models can be generalized to reflect relationships among object sets of size greater than two, as occurs when the context size is larger than one.

Locally Linear Embedding (LLE) [37], [36] is a non-linear dimensionality reduction method which also yields low-dimensional object representations. This is accomplished by first learning a reconstruction of each point in the data using a weighted sum of the positions of its nearest neighbors. Then, each point is mapped non-linearly to the low-dimensional space in such a way that the new representation is close to the weighted sum of its original neighbors' new representations. The weights are fixed in this step while the new representations of the objects are freely learned parameters of the model, much like the embedding functions in our work. In this way, a non-linear mapping of all the objects is learned that preserves the original local relationships among nearest neighbors in the original space. While this method yields a low-dimensional representation of the data, it requires an existing vector space representation of the data, unlike embedding models which map object IDs directly to a new space.

One of the most similar models to ours appears in [14]. In this work, the authors use a specific case of our model to relate pairs of heterogeneous objects (e.g. authors and words) using a Euclidean scoring function and generative formulation. Despite this, the primary difference between this and our work is that the authors only consider a subset of the possible models in our work by considering only pairwise relationships.

The skip-gram model [28], [29] learns representations of words in text to try to predict which target words will occur before and after a context word in

the text. This yields an efficient training method and semantically meaningful representations of words in text. However, the authors again mainly consider pairwise relationships. Furthermore, in this case the objects are homogenous in nature – the authors only consider relationships between pairs of words specifically. The defined model is also more generally mostly concerned with types of objects that occur in sequence – such as words in text.

2.4.2 Music Information Retrieval

The field of Music Information Retrieval, or MIR, is concerned with the application of computational methods, in particular machine learning and information retrieval, to music-related domains. Our work attacks problems in this domain related primarily to recommendation systems and data analysis. First, we explore how to use embedding methods to generate good music playlists without relying on audio features. The reliance of our method only on observed co-occurrence between songs is an advantage of our method, since music is a highly social and cultural phenomenon and even songs that appear similar in audio may appeal to very different groups of people.

The problem of playlist generation is related to that of recommendation systems. The primary difference is that playlist generation is particularly concerned with the sequential coherence of consecutive songs. Indeed, in [2], the authors use recommender systems to generate playlists. However, this work does not incorporate a notion of sequential coherence for which we wish to optimize.

In [24], the authors investigate generating steerable playlists using audio

similarity between songs. A user of this system can then influence the trajectory of the playlist by selecting a number of descriptive tags (genre descriptors like *rock* or *jazz*, instrumental descriptors like *violin* or *guitar*, mood descriptors like *chill*, etc.). The system then tries to steer the playlist towards songs that are described by the selected tags. However, in this system the notion of song similarity is based on the audio signals of the songs. In addition, the tags here are generated by a content-based autotagger, which again takes audio signal as an input.

The work in [26] proposes considering the playlisting problem as an analog to language modeling in natural language processing. Namely, the authors suggest considering playlists containing sequences of discrete songs as parallels to sentences containing sequences of discrete words. Using this analogy, the authors establish the use of the log-likelihood as a performance metric for playlisting methods (based on historical playlist data). The authors evaluate playlist generation approaches based on audio similarity, tag similarity, familiarity (or popularity), random shuffle, and an optimized combination of all these approaches. The audio and tag models perform poorly on their own and are outclassed by even the random shuffle strategy. Furthermore, the optimized combination method, on average, places low weight (9%) on the audio distribution, suggesting that audio similarity alone is not sufficient for good playlists. Our playlisting work can be seen as building on these results – the non-linear embedding functions we learn are able to arrange songs due to similarities in historical playlist data, which may result from social or cultural phenomena outside of audio.

In work that is complementary to our approach, [27] explores the use of

weighted hypergraphs for playlist generation. The authors define a model based on the assumption that users, when selecting songs for a playlist, first narrows the search to a specific subset of songs based on a single genre or some other characteristic. Then, the user chooses a song from this subset uniformly at random. The subsequent song is chosen by first selecting another characteristic group (e.g. another genre descriptor) containing the current song, and again choosing a song from that group uniformly at random. The authors encode these assumptions using weighted hypergraphs and include features for edges based on many sources of information, including audio, collaborative filtering similarities, social tags, and more. Playlists are then generated by a weighted random walk on the hypergraph, where a song is chosen uniformly at random from each selected hypergraph edge. On one hand, our model offers a simplified and more easily understood notion of song similarity than is given by this hypergraph model – in the latter, two songs are similar if they have many groups in common with high weight, whereas similarity in our models is defined simply by Euclidean distance. On the other hand, the uniform choice of a song given a hypergraph edge could be replaced by some other distribution. In particular, it would be possible to incorporate embedding models like ours into this model in order to improve performance. This would also interestingly yield a fine-grained metric space reflecting similarities among songs in, e.g., a specific genre or audio cluster.

A number of other authors use content-based approaches for some of the tasks we consider. The authors in [21] use audio similarity as well as text mined from web searches for artists to generate playlists. Likewise, [23] considers generating playlists by picking the nearest neighbors to a seed song in terms of similarity in certain audio features. In the realm of data analysis, [18] applies

locally linear embeddings to speech and music data for the purpose of visualization. This approach finds a low-dimensional embedding for existing acoustic features. In [39], the authors use audio features in topic models with temporal dynamics to explore the concept of musical influence over time. However, all of these works are content-based and thus rely on the use of acoustic features, unlike our methods.

Other methods consider creating spaces using existing similarity metrics, such as Platt's work [35]. In this particular application, a music similarity graph is embedded into this new space in order to generalize pairwise simliarities between pairs of songs. In other work, Gleich et al [13] creates a graph based on collaborative filtering similarity between pairs of artists, and then uses a dimensionality reduction method to embed the graph in three dimensions. However, unlike our methods, the representations discovered in these works lack a probabilistic interpretation or a generative component.

Further works investigating methods related to recommender systems in MIR include the work of Aizenberg et al [1]. In this work, the authors gather a large amount of data from Internet radio stations in order to build a recommender system. This work is also notable in that the authors use an embedding method much like ours to create a generative model of artist plays by radio stations. However, the end goal is the recommender system itself – no attention is paid here to sequential playlist generation or visual data analysis tasks. Additionally, Dror et al [11] present a temporal collaborative filtering system for music, but too lacks any data analysis component.

Part of the work we present focuses on geographical analysis of music listening behavior. Our analyses here build on the work of Hauger, Schedl, et

al in [16], [38], [17]. In these works, the authors mine large collections of microblog data from Twitter and analyzing the geographical patterns that arise. The analyses presented in these works are interesting and novel, and in our work we present additional analyses of the data in [16] in particular using embedding methods. The advantage of our embedding methods here partly stems from their ability to generalize similarities between artists and between cities. This enables the methods to show us a more coherent picture of similarities and differences between different geographical areas, which we use to investigate similarities on a city to city level.

In work concurrent to ours, Jun et al [20] mine a dataset from Twitter and also use this to model the similarities between geographical entities. The model they develop uses a two-dimensional map which is learned so that euclidean distance directly approximates a notion of genre distance. However, the authors only consider two-dimensional models, whereas in our geographical work we find that many of the most subtle and interesting patterns arise through the application of higher dimensional models. Another work focusing on the geography of music is the work of Knopke [22]. This work considers a web crawl for sounds which attempts to disambiguate from where each sound originated geographically. In terms of data analysis, the results found here are largely preliminary results, and the main contribution of the work is the data set itself. This data set could be an interesting future target of investigation for our methods.

2.4.3 NLP and Information Retrieval

Embedding methods have also found application in the domains of natural language processing and information retrieval. The work of Maron et al [25] uses a very similar model to ours in order to learn representations of words in text. These representations are then clustered in order to perform unsupervised part-of-speech tagging. There has also been extensive work in embeddings for language modeling, especially the work of Bengio et al [4, 5, 3]. In these works, the authors employ a neural probabilistic language model in order to model sequences of words. These models are also similar in flavor to ours, however, the scoring functions involved assume the form of a neural net. This added layer of complexity harms the interpretability of the models in comparison to our own.

Finally, the work of Weston et al [40, 41] learns embeddings that are optimized for a ranking task. In particular, the models are learned so that for any given query object, the inner product between its representation and that of a candidate result object is large if that result is highly relevant to that query, and the ranking metric precision at k is the objective function. These works describe an application to image annotation [41] as well as to music information retrieval [40]. The primary distinction between these models and ours is the focus on a ranking problem. Whereas these models are optimized for a retrieval task, our models are optimized for generative modeling, which is an important characteristic for playlist modeling in particular.

2.4.4 Efficient Training

As mentioned earlier, the normalizing factor known as the partition function poses a challenge in terms of efficiency and scalability for training our models. In the next chapter, we will discuss our own conceptually simple and effective approach for overcoming this limitation, but a number of previous approaches exist.

In the work of Bengio et al [5], the authors apply a technique known as *importance sampling* to speed up training. In this approach, instead of sampling from the computationally expensive model distribution in the process of calculating the update, samples are taken from an easy-to-compute proposal distribution, such as the unigram distribution. The samples must be re-weighted in order to correctly compute the update, and the estimate involved is biased. This method offers a significant speed-up over exact calculation of the partition function. However, as the model distribution diverges from the proposal distribution over the course of training, the number of samples required becomes large and updates again become expensive to compute.

To overcome this, the authors proposed a scheme called *adaptive importance sampling* [4]. In this method, a weighted combination of n-gram distributions of various orders is tracked over the course of training. The weights in the combination are updated in an attempt to keep the mixture close to the model distribution. This alleviates the problems of the growth of the number of samples, but the resulting method is conceptually complicated.

The work of Mnih et al [31, 30] proposes the use of a method called *noise-contrastive estimation*, or NCE, to speed up training. This method was originally

developed in the work of Gutmann, et al [15], and attempts to separate noise data and true data samples through the use of nonlinear logistic regression. The end result is a fairly simple method with a hyper-parameter k for the number of noise samples performed per update.

Finally, Chen et al [7] describe an approximate, distributed, embarrassingly parallel method for training embedding models. The idea here is to partition the items of the space and create almost disjoint sub-spaces for each set in the partition. Each space contains special points (called portals) for each of the other spaces which, in the generative model, signify a jump to the other space. The model only approximates the quality of the original model, and the approximation degrades as the number of clusters increases, but the approximation is good overall. In addition, the runtime ideally decreases by a factor of $\frac{1}{k^2}$, where k is the number of sets in the partition, due to the quadratic nature of the partition function calculation. Furthermore, since the partition function is calculated exactly in each subspace, it seems possible to combine this method with another speed-up method such as NCE or our own. This would provide the potential for distributed computation of extremely large models, plus a very significant speed-up for the training of each individual subspace.

CHAPTER 3

EMBEDDING SONGS FOR PLAYLIST GENERATION

3.1 Playlist Modeling

A music consumer can store thousands of songs on his or her computer, portable music player, or smart phone. In addition, when using a cloud-based service like Rhapsody or Spotify, the consumer has instant on-demand access to millions of songs. This has created substantial interest in automatic playlist algorithms that can help consumers explore large collections of music. Companies like Apple and Pandora have developed successful commercial playlist algorithms, but relatively little is known about how these algorithms work and how well they perform in rigorous evaluations.

Despite the large commercial demand, comparably little scholarly work has been done on automated methods for playlist generation (e.g., [35, 13, 24, 26]), and the results to date indicate that it is far from trivial to operationally define what makes a playlist coherent. The most comprehensive study prior to our work was done by [26]. Working under a model where a coherent playlist is defined by a Markov chain with transition probabilities reflecting similarity of songs, they find that neither audio-signal similarity nor social-tag-based similarity naturally reflect manually constructed playlists.

In this work, we therefore take an approach to playlist prediction that does not rely on content-based features. Playlists are treated as Markov chains in some latent space, and we use embedding methods to learn representations of the songs in this space. Training data for the algorithm consists of existing

playlists, which are widely available on the web. Unlike other collaborative filtering approaches to music recommendation like [35, 13, 40], ours is among the first (also see [1]) to directly model the sequential and directed nature of playlists, and that includes the ability to sample playlists in a well-founded and efficient way.

In empirical evaluations, embedding-based sequence models substantially outperform traditional n-gram sequence modeling methods from natural language processing. Unlike such methods, embedding methods do not treat sequence elements as atomic units without metric properties. Instead, they provide a generalizing representation of songs in Euclidean space.

3.2 Markov Chains in Embedding Space

Playlist modeling involves modeling sequences of songs such that generated sequences reflect good, coherent, natural playlists. Therefore, in this chapter, we develop an application of embedding methods that uses the Markov property to model sequences through the use of a latent space. In this scenario, our vocabulary V consists of a set of songs, which we will denote $S = \{s_1, ..., s_{|S|}\}$. Our training data consists of a set of playlists p, which are ordered sequences of songs. We will adopt the notation p_i for the ith element of the playlist p, such that $p = (p_1, ..., p_{k_p})$. Our modeling task is to estimate a distribution $P(p)$ of coherent playlists, which we will accomplish by first assuming that the distribution follows the Markov property. That is, the probability of p decomposes into the product of transition probabilities $P(p_i \mid p_{i-1})$. We further assume that the seed song is given for each playlist.

Under these assumptions, the notion of context becomes the single previously played song s, and any collection of playlists decomposes to a collection of bigrams (s, s') of consecutive songs, which constitute the context-object pairs for our training algorithm. In other words, a playlist $p = (p_1, ..., p_{k_p})$ decomposes into the set of bigrams $\{(p_1, p_2), ..., (p_{k_p-1}, p_{k_p})\}$. The data set becomes the union of these decomposition sets over all playlists p in the data. Training then consists of repeatedly sampling a song pair (s, s') from the data distribution D (which in this case is the bigram distribution over songs in the data) and a song s'' uniformly at random. Then, $X(s)$ and $Y(s')$ are drawn closer together by the gradient step while $X(s)$ and $Y(s'')$ are pushed away from each other. In this way, songs that frequently occur together in the playlist data are drawn close to each other, while songs that rarely co-occur are pushed far from each other. At convergence, the resulting space clusters together songs that belong in playlists together. A random walk over this space will then yield a good, coherent playlist.

We will consider two variants of the emedding model in this section. On one hand, we consider the case where $X(\cdot) = Y(\cdot)$. In this scenario, known as the *single point model*, each song is given the same representation in embedding space regardless of whether it occurs as the context (the song that just played) or the target (the song that is being considered for the next play). On the other hand, we also examine the effect of allowing $X(\cdot)$ and $Y(\cdot)$ to differ. In this case, which we refer to as the *dual point model*, each song s has totally independent points $X(s)$ and $Y(s)$ as context and target representations, respectively. Under these circumstances, $X(s)$ can be thought of as an *exit point* while $Y(s)$ can be considered an *entry point* – two songs are a good fit for each other if the exit point of the first is close to the entry point of the second. This allows the model to incorporate an additional degree of asymmetry, which may prove useful in

sequence modeling.

In both of the models we consider, the model benefits greatly from the transfer of information among different songs. Songs with similar empirical conditional distributions $P(s'|s, D)$ will end up close together in the embedding space. Since they are nearby in the metric space, they will have similar model distributions both on songs that were observed as successors in the training data and on those songs which were never seen played after the song in question. Similarly to how collaborative filtering recommenders are able to fill in gaps in knowledge by considering users with similar tastes in music, our model is able to better determine when a bigram that was never observed in the data is nonetheless probably a good bigram. This gives the model a great advantage over empirical bigram models for the task, since the latter must employ smoothing in order to assign a non-zero probability to every bigram.

3.3 Baselines From Language Modeling

The baselines we use are motivated by techniques for language modeling in natural language processing. In NLP, language models are employed in order to estimate the likelihood of a given sequence of words appearing in text. This forms a clear parallel to our playlist generation task: playlists are essentially sentences of songs.

We define the following baseline models:

- **Uniform Distribution:** in this case, at each step in the playlist generation, the probability of any song $s \in S$ being selected is exactly $\frac{1}{|S|}$.

- **Unigram Distribution:** this distribution draws from songs proportionally to the number of times each song has appeared in the training data. That is, if N_s is the number of occurrences of song s in the data and N is the total number of track plays in the data, then under the unigram model, $P(s|s') = \frac{N_s}{N}$ regardless of the previously played song s'.
- **Bigram Distribution:** given a previously played song s', this distribution draws a song s proportional to the number of times s occurred after s' in the training data. That is, if $N_{(s',s)}$ is the number of times song s occurred after song s', and $B_{s'} = \sum_s N_{(s',s)}$, then $P(s|s') = \frac{N_{(s',s)}}{B_{s'}}$.

In the case of the bigram distribution, the issue arises that bigrams which were unobserved in training are assigned zero probability. To solve this, a number of smoothing techniques exist in the language modeling literature for intelligently assigning some of the probability mass from observed bigrams to the unobserved ones. Many of these interpolate the bigram model with a unigram model by relying on an assumption that with some small probability, the bigram model should instead use a lower-order unigram model to generate the next word. A study of several such techniques is given in the work of Chen et al [8].

For our work, we use Witten-Bell smoothing to estimate the probabilities of unobserved bigrams. Under this scheme, the probability of falling back to the lower-order model is a function of how many distinct tokens follow a given context relative to the total count of bigrams beginning with that context. In this way, contexts with many distinct but rarely occurring successors place heavy weight on the lower-order model, since there is intuitively a high probability of these contexts encountering unseen successors. On the other hand, contexts

with few distinct successors but a large number of total observations have a low probability of encountering an unobserved successor, and therefore are assigned a low weight for the lower-order model.

3.4 Historical Playlist Data

Our playlist algorithms require historical playlist data for training. This requires no audio or other content-based data about the songs involved, only a set of playlists consisting of sequences of song IDs. Such data is widely available on the web. For example, the on-demand music streaming service Spotify contains a large number of public playlists The mix and podcast listening website Mixcloud hosts hand-designed mixes, many of which have track labels which are accessible through their API. Art of the Mix also hosts handcrafted playlists.

For this work, we obtained radio playlists from *Yes.com*, a website which logs radio playlists from stations within the US. We crawled playlists from stations of all genres from December, 2010 to May, 2011. This resulted in around 1.9 million bigrams from 75,262 distinct songs. In our experiments, we consider this original data set, known as *yes_complete*, as well as two pruned versions of the data. In the larger pruned version, *yes_big*, we discard all songs with fewer than 5 occurrences (as well as all corresponding transitions, splitting playlists into two as necessary). In the smaller pruned version, *yes_small*, we remove songs with fewer than 20 occurrences. In the end, *yes_big* contains about 1.8 million bigrams and 9,775 distinct songs, while *yes_small* contains about 1.3 million bigrams and 3,168 distinct songs.

We further divide each data set into training, validation, and test sets. This

split is performed on the basis of the playlists themselves. Specifically, for the training set, a random subset of playlists is selected so that the total number of transitions is about 80%. This is performed by placing the playlists in a random order (without disturbing the internal order of any individual playlist), and selecting the first n playlists that constitute at least 80% of the transitions. The same technique is performed to select the next 10% of the transitions for the validation set, with the remaining 10% or so being assigned to the test set. The validation sets are used for early stopping during training as well as for training parameter selection, and all reported performance figures are given for the test set.

3.5 Experiments

First, note that our model requires a dimensionality d as a parameter, and the training routine requires a learning rate η to be specified as well. The choice of dimensionality partly depends on the desired application. When $d = 2$, we gain the advantage that the model is easy to inspect visually. This can help us immediately see patterns in the data that might not be obvious otherwise. However, higher dimensionalities allow the model to capture more of the signal in the data. This leads to higher performance in modeling the data, so for the actual generative task of producing playlists, a larger setting of d is preferred. In order to explore both ends of this spectrum, we train models for each d in $\{2, 10, 50, 100\}$ and indicate performance plots for each of these.

The choice of η, on the other hand, has little consequence outside of the quality of the resulting model. For this reason, we generally only display re-

sults for the best-performing η we find. The models are trained for each η in $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$, and the best model is chosen based on the calculated exact log-likelihood (that is, the original objective $LL(D)$, not the bound $LL_b(D)$) on the validation set.

3.5.1 Qualitative Results

In Figures 3.1 and 3.2, we show visual plots of 2-dimensional single-point models trained on the *yes_small* and *yes_big* datasets respectively. This gives us a qualitative impression of the nature of the embeddings produced by our method. Songs from several reference artists of different genres are plotted in order to give more insight into the structure of the space.

First, it is interesting to note that songs by the same artist cluster tightly, even though our model has no direct knowledge of which artist performed a song. Second, logical connections among different genres are well-represented in the space. For example, consider the positions of songs from Michael Jackson, T.I., and Lady Gaga. Pop songs from Michael Jackson could easily transition to the more electronic and dance pop style of Lady Gaga. Lady Gaga's songs, in turn, could make good transitions to some of the more dance-oriented songs (mainly collaborations with other artists) of the rap artist T.I., which could easily form a gateway to other hip hop artists.

While these visualizations provide interesting qualitative insights, we now provide a quantitative evaluation of model quality based on predictive power.

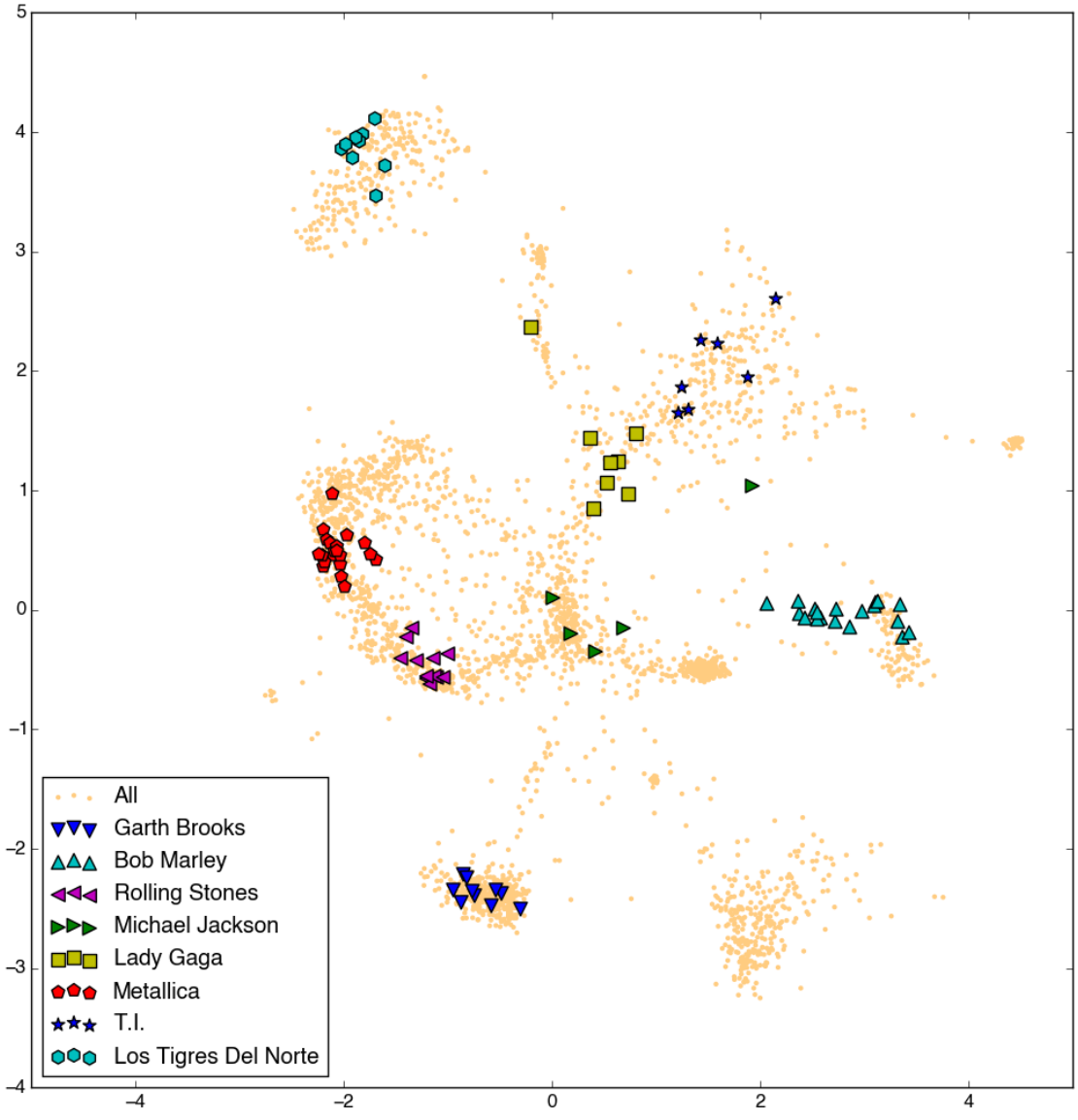

Figure 3.1: A 2-dimensional plot of a model trained on the *yes_small* data set with key artists highlighted.

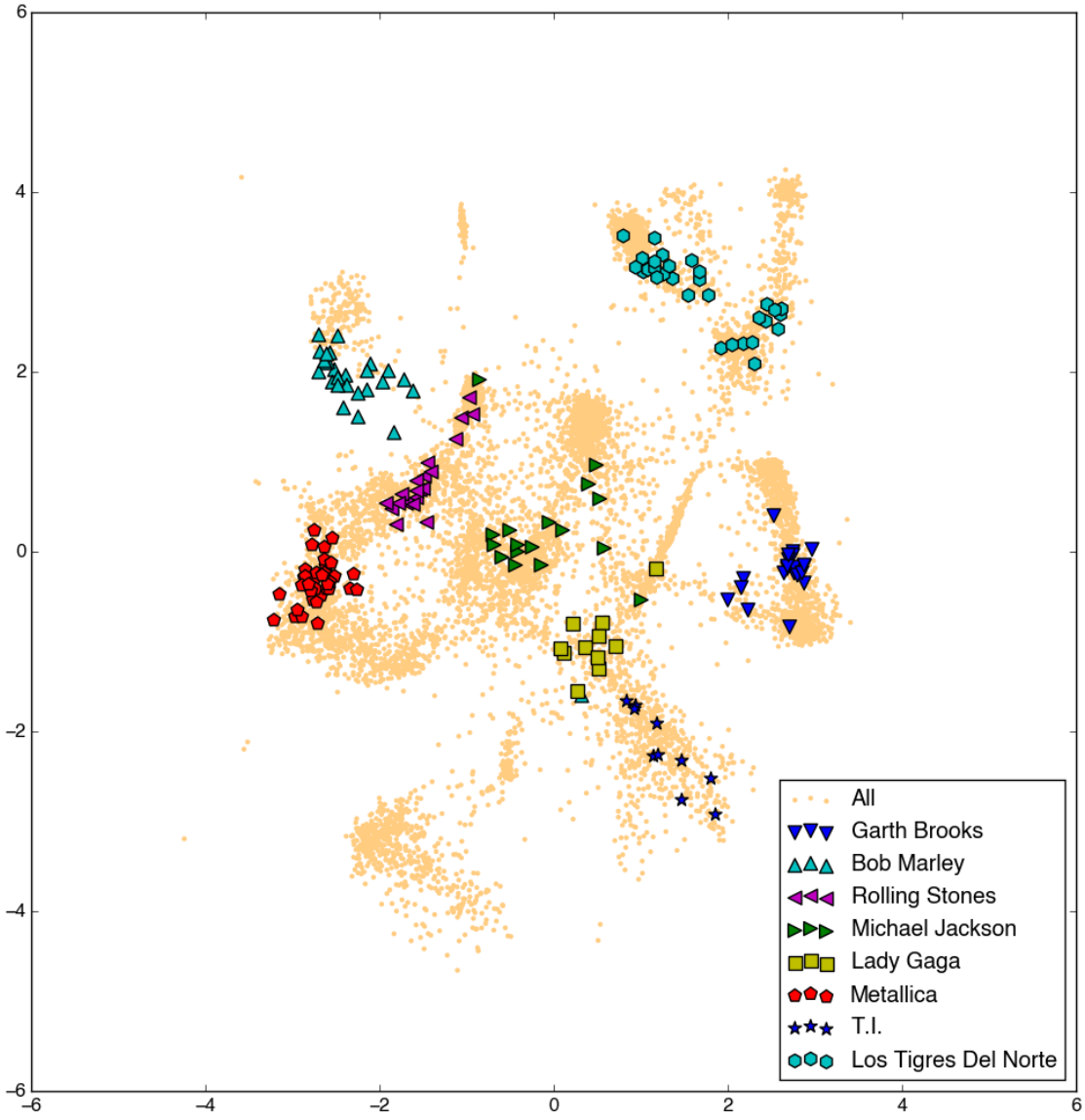

Figure 3.2: A 2-dimensional plot of a model trained on the *yes_big* data set with key artists highlighted.

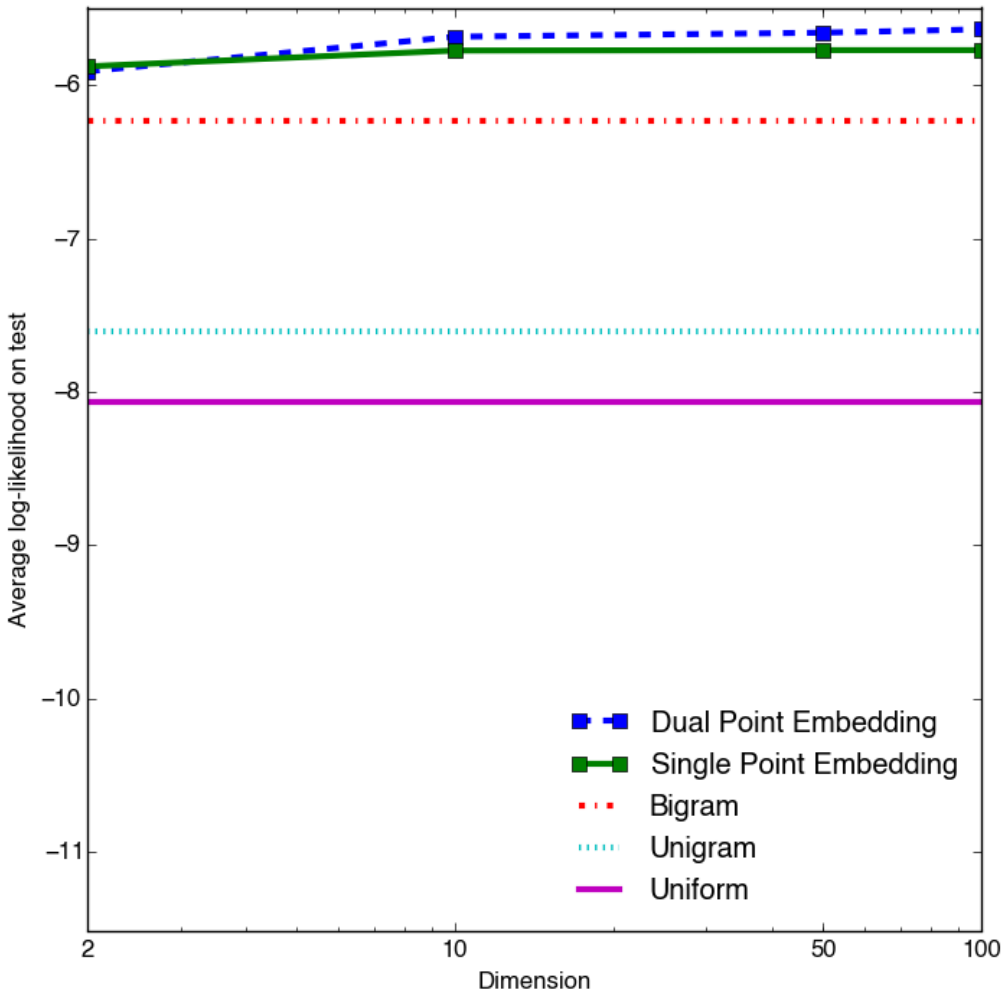

Figure 3.3: Comparison of embedding methods against baselines on *yes_small*.

3.5.2 Comparison with Baselines

In Figures 3.3 and 3.4, we show the performance of embedding methods against the sequence-modeling baselines we have defined. In each case, the uniform baseline performs the worst as expected, with the unigram distribution seeing a large gain over uniform, and the bigram distribution seeing an even more significant gain. But for each data set, we also see a very large gain in likelihood

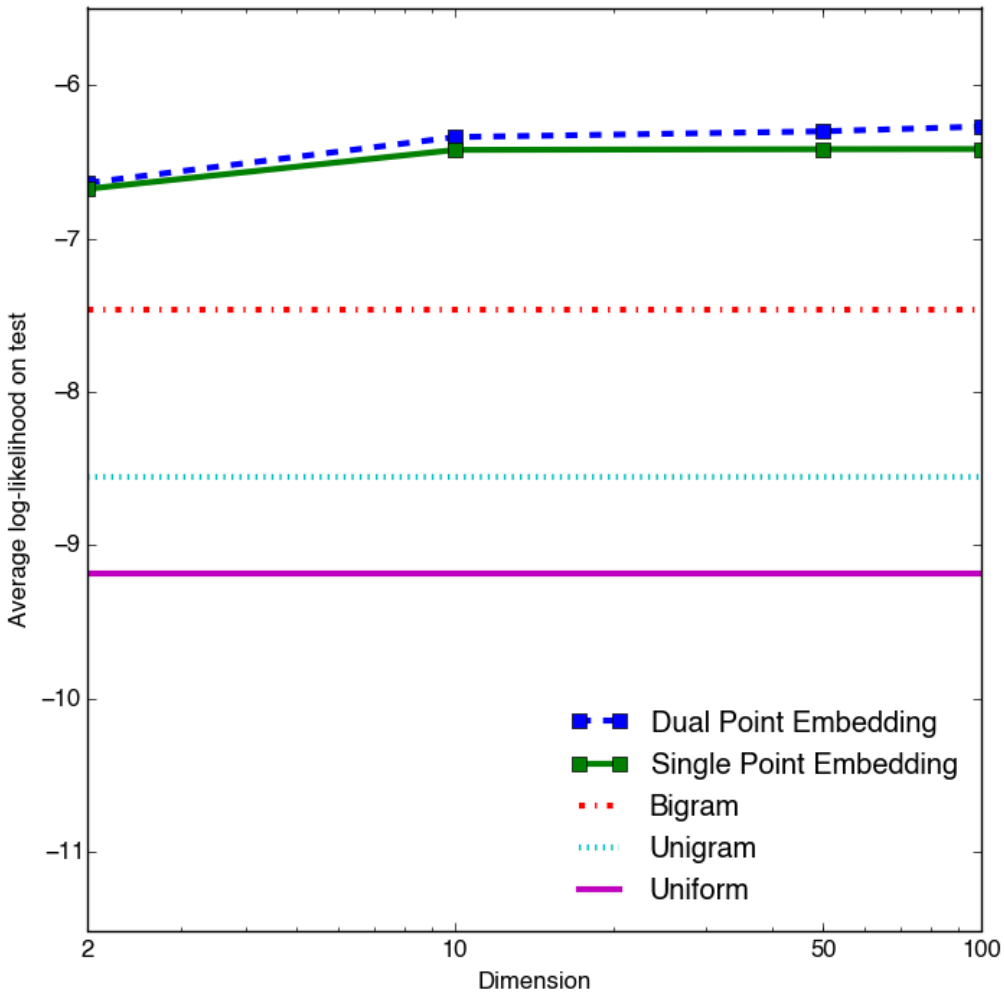

Figure 3.4: Comparison of embedding methods against baselines on *yes_big*.

for our method against any of the baselines, regardless of dimension. The gain of adding more parameters to the model by increasing dimensionality levels out very quickly – a 10-dimensional model does not perform much worse than a 100-dimensional model. Additionally, we see that the dual point model obtains a substantial gain in performance over the single point model, albeit not

as large a gain as either of these over the baselines. [1] It is also interesting to note that the difference in performance between our models and the baselines is more pronounced for *yes_big* than *yes_small*. This is likely due to the increased sparsity (in terms of the ratio of observed bigrams to the size of the vocabulary) of the *yes_big* data set. Namely, since fewer bigrams are observed for the given songs, and since our model's primary advantage over the bigram model is its improved ability to estimate the probability of unobserved bigrams, the ultimate difference in performance is greater for this sparser data set.

This hypothesis can be justified by observing the results in Figures 3.5 and 3.6. In these figures, we consider the test bigrams on the basis of how frequently they were observed in the training data. The line plots indicate the performance of the two models in terms of log-likelihood when restricted to bigrams that were observed in the training data with the frequency indicated on the x axis. The bars indicate the proportion of the test bigrams which were observed with that frequency in the training set. For the embedding models, we selected the training parameters (learning rate, dimensionality, and the use of a single point vs. dual point model) according to performance on the validation set. We can see that in the *yes_small* data set, for bigrams that were observed even once in the training set, the embedding model performs almost identically to the bigram model. However, with respect to bigrams that were not present in the training set (about 22% of the test bigrams), the embedding model's performance is vastly superior to that of the bigram model. The results are similar for the *yes_big* data set. In this case, on bigrams that were observed from 1 to

[1] The results on the dual point model differ from those published in [6] on a similar dataset. In that work, the performance of the dual point model degraded severely at larger dimensions. We believe this to be due to bad local optima found by the training method used in that work. Specifically, models were trained with exact calculation of the partition function, and we speculate that our approximate method is more robust to local optima.

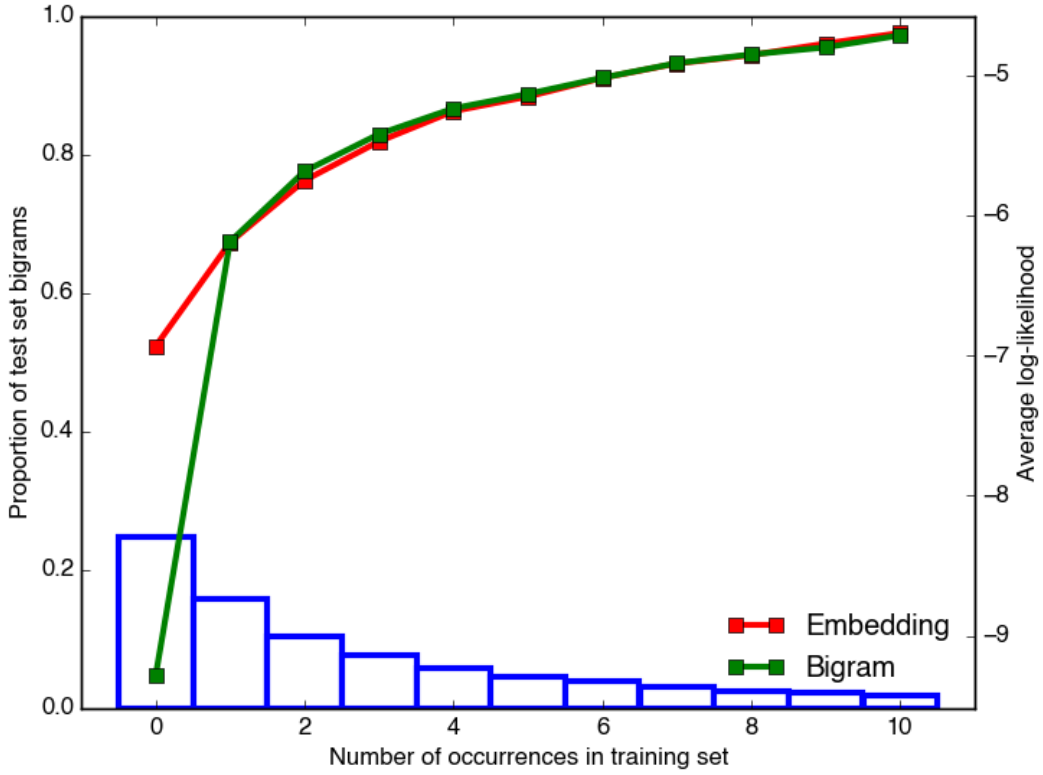

Figure 3.5: Comparison of the performance of a bigram model and an embedding model on bigrams of varying observed frequencies in the training data for *yes_small*. The bar chart presents the proportion of test bigrams which were observed that frequently in the training set.

about 4 times, the bigram model offers slightly improved performance. However, for higher frequencies the performance is again identical between the two models, and for bigrams that were never observed in training, the embedding model is again far better. For this data set, the proportion of test bigrams that were unobserved in training is even higher at almost 45%.

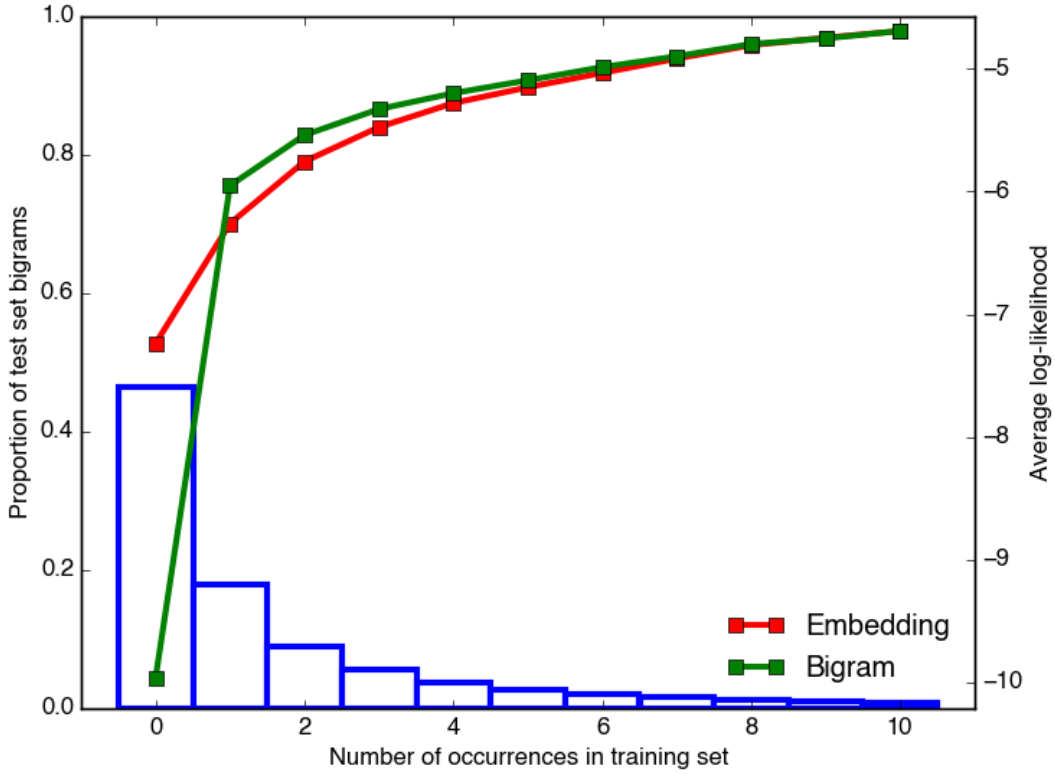

Figure 3.6: Comparison of the performance of a bigram model and an embedding model on bigrams of varying observed frequencies in the training data for *yes_big*. The bar chart presents the proportion of test bigrams which were observed that frequently in the training set.

3.5.3 Approximation Quality

In Chapter 2, we detailed an approximate training routine for scalable learning. This routine involves maximizing a lower bound on the true objective, which allows us to estimate the expectation of the partition function $Z(c)$ over all contexts c proportional to their occurrences in the data. By using this estimate, we avoid calculating the exact partition function, which in this application would require a sum over all the songs for each context song. In turn, this calculation

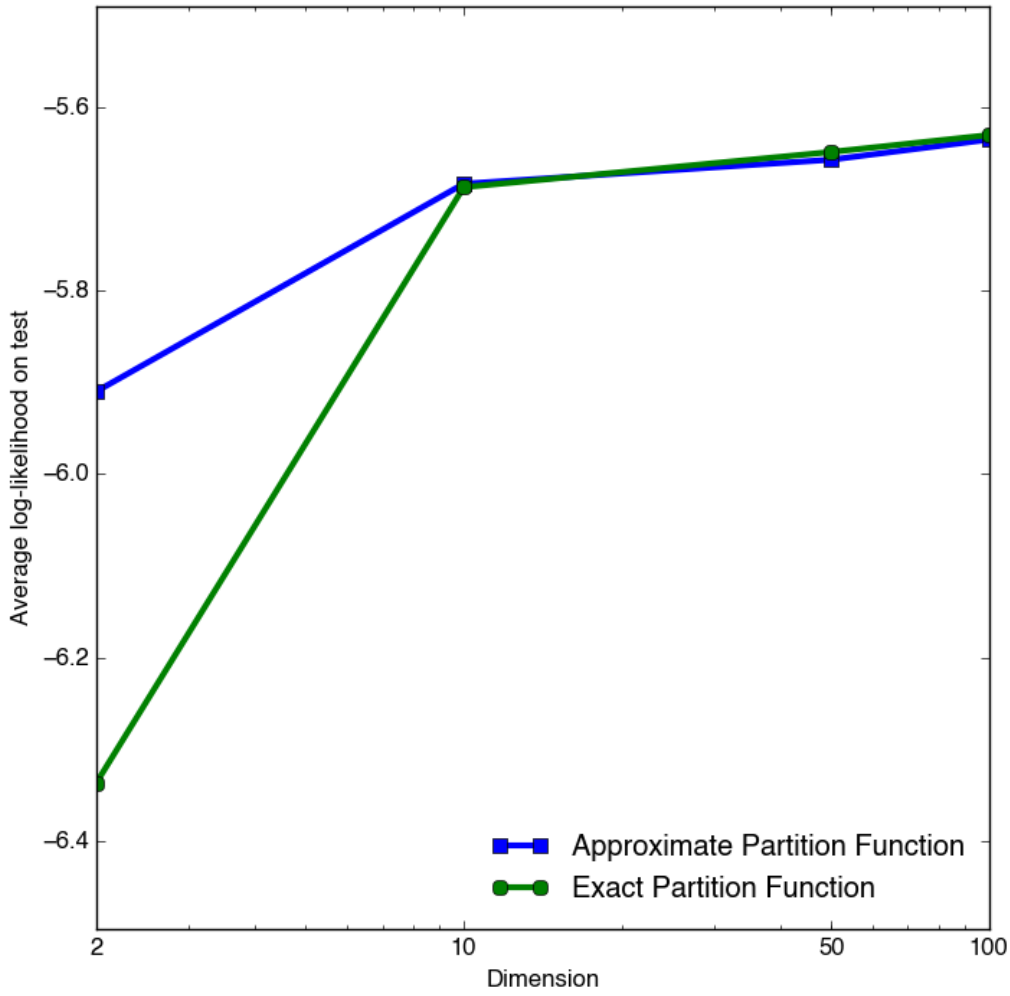

Figure 3.7: Comparison on the *yes_small* data set of the approximate training algorithm detailed in Chapter 2 with a method that uses exact calculation of the partition function. For each data point, the test likelihood is shown for the model which performed optimally on the validation set. The exact method also optimizes the exact objective while the approximate method maximizes a lower bound on the true objective.

would lead to an update complexity which is quadratic in the number of songs in the data.

In Figure 3.7, we compare the performance of this approximate method to that of the exact method. Due to the expensive computation required for the exact method, we compare only on the *yes_small* data set. Furthermore, all models in this experiment are dual point models. We choose the models that performed optimally on the validation set and display the results for these on the test set. We can see that in the case of two-dimensional models, the approximate method actually far outperforms the exact method. This could be due to the lack of concavity in the training problem. If there are many local optima in the optimization surface, the noisy nature of the approximation could actually help the training process avoid getting stuck in them. For higher dimensional models, the difference in performance of the two training algorithms is negligible.

3.5.4 Comparison of Training Algorithm to Noise-Contrastive Estimation

Of the existing training approaches described in Section 2.4, the most efficient existing method seems to be the *Noise-Contrastive Estimation* (NCE) approach described in the work of Gutmann et al [15]. This method has been applied to models similar to ours in the work of Mnih et al [31, 30]. NCE essentially considers a regression problem that attempts to distinguish data samples from noise samples. The resulting method takes a noise distribution and a number of noise samples k per update as input, and is similar to our training method in terms of the structure of the training process. Algorithmically, the main differ-

ence is that the repulsive portion of the update is applied to each of the noise samples drawn, and all of the updates are rescaled by quantities related to the posterior probability that a sample was drawn from the noise distribution or the data distribution. Since k repulsive updates are applied for each iteration of the training process in NCE, as opposed to a single repulsive step in our method, the complexity of one update is increased by a multiplicative factor of k. This may be offset, however, by faster convergence in terms of the number of training iterations.

Intuitively, two factors in particular may influence the effectiveness of our method. First, as the dimension of the model increases, the Euclidean distance between pairs of objects in a structured space increases. All other things equal, this would cause the magnitude of the partition function to decrease. Since the log function in our objective is closer to linear for larger values, this would in turn cause our Jensen's inequality bound to become less tight, potentially harming the objective that can be achieved with our method. Second, as a space becomes less sparse in terms of the number of objects it contains – for example, in larger data sets – the scale of the partition function may grow, causing the bound to become tighter.

In this experiment, we wish to compare our training method to NCE in terms of the speed of convergence and the quality of the resulting model. To this end, we trained dual point models with bias terms on *yes_small* and *yes_big* using both our bounded likelihood method and the NCE method. In each case, we determined convergence by calculating the average log-likelihood on the validation set and ending training when this stopped improving. We validated across dimensions 2 and 100 and each step size η in $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$.

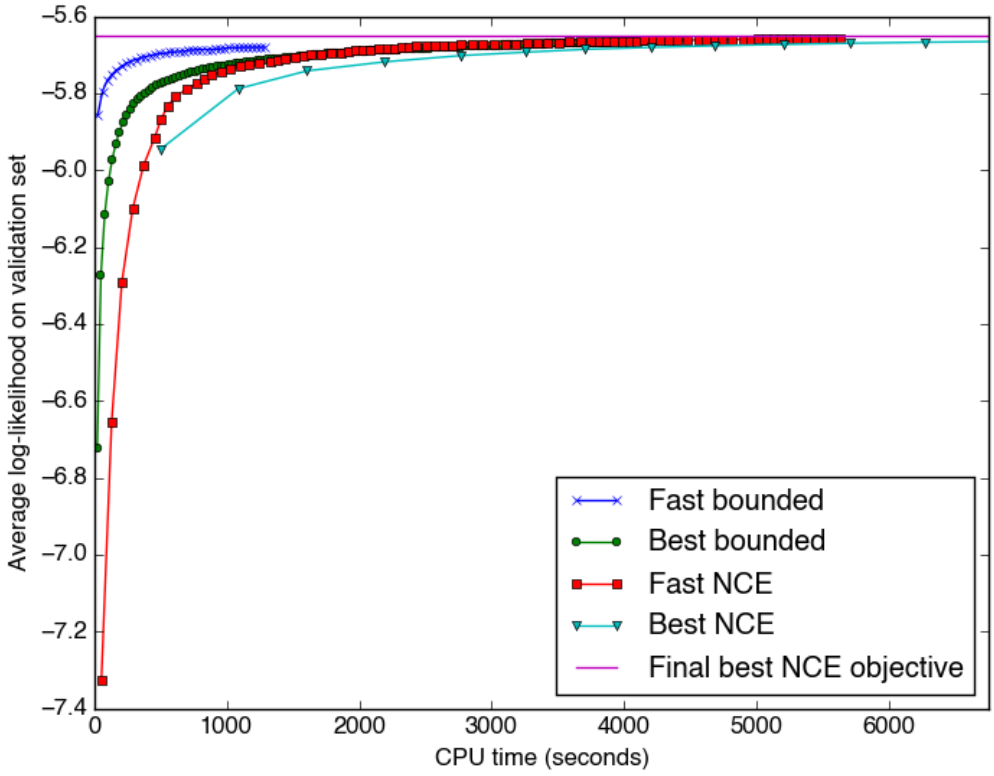

Figure 3.8: Comparison of our likelihood bounding training method to the noise-contrastive estimation approach on the *yes_small* data set. *Best bounded* and *Best NCE* indicate the bounded likelihood method and NCE method (respectively) which converged to the best validation objective. *Fast bounded* indicates the bounded likelihood method which uses the larger step size from the best NCE method in order to converge more quickly. *Fast NCE* indicates the fastest-converging NCE method which still matches the fast bounded method in the obtained validation objective at convergence.

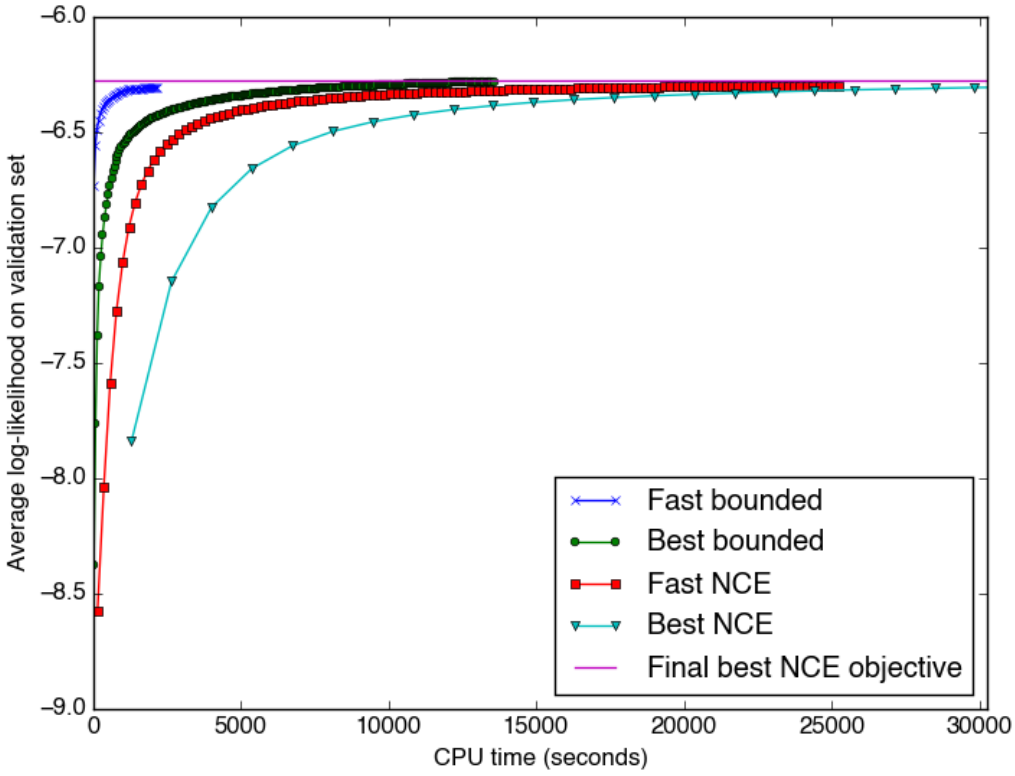

Figure 3.9: Comparison of our likelihood bounding training method to the noise-contrastive estimation approach on the *yes_big* data set. *Best bounded* and *Best NCE* indicate the bounded likelihood method and NCE method (respectively) which converged to the best validation objective. *Fast bounded* indicates the bounded likelihood method which uses the larger step size from the best NCE method in order to converge more quickly. *Fast NCE* indicates the fastest-converging NCE method which still matches the fast bounded method in the obtained validation objective at convergence.

The NCE method has two additional parameters: the number of noise samples k and the choice of noise distribution. We considered values of k in $\{1, 10, 50, 100\}$. As for the noise distribution, in [31], the authors considered the uniform distribution and the unigram distribution, a choice which we mirror here.

In the case of $d = 2$, our method matched the NCE method in terms of obtained validation and test objectives on *yes_small*. The optimal model for both cases obtained a likelihood of -5.90, and the best-performing model on the validation set obtained a test likelihood of -5.89. However, our method converged in 255 seconds, whereas the NCE method converged in 6,830 seconds – in other words, our method offered a 26 time speedup over NCE. The optimal NCE method in this case used $k = 100$. By using smaller values of k, the method can sacrifice some modeling performance for a more reasonable convergence time. For example, the second and third-best NCE models in terms of validation objective used $k = 50$ and $k = 10$, achieved validation objectives of -5.94 and -5.97 and test objectives of -5.92 and -5.95, and converged in 2,670 seconds and 2,326 seconds respectively. However, despite a significant degradation of performance, even these training routines took roughly ten times as long to converge as did our method.

The results for $d = 100$ are shown in Figures 3.8 and 3.9 for *yes_small* and *yes_big* respectively. In these figures, we display the validation objective with respect to training CPU time in seconds. Two of the four results in each figure correspond to the best NCE and bounded likelihood models in terms of validation objective. We also show a faster parameter setting for each method in each figure. Namely, training convergence time is greatly influenced by the choice of learning rate η – larger learning rates lead to faster convergence. In the case of

NCE, the optimal learning rate is 0.05 for each data set, while the optimal choice is 0.01 for the bounded likelihood method. However, by increasing this learning rate slightly to 0.05 for the bounded likelihood method, convergence time can be greatly reduced at only a small cost to validation objective. The fast bounded method curve in each figure reflects this choice. For NCE, we attempted to fairly choose a fast NCE method by choosing the fastest method which performed at least as well as the model resulting from the fast bounded curve.

Note that in each figure, the x axis is constrained to end soon after the convergence of the fast NCE method in order to improve legibility. The actual convergence times of the truncated best NCE curves were 19,394 seconds in the case of *yes_small* and 85,223 seconds in the case of *yes_big*. For *yes_small*, the best bounded method achieves a validation and test objective of -5.66 in 4,167 seconds of CPU time. The best NCE method achieves a slightly improved objective of -5.65 on the validation set and -5.63 on the test set, but takes 4.6 times as long to converge. The fast bounded method obtains validation and test objectives of -5.68 and -5.66 with only 1,296 seconds of computation, and the fast NCE method obtains validation and test objectives of -5.65 and -5.64 with 5,634 seconds of computation. This means that the fast NCE method still requires 35% more time to converge than the best bounded method, while the fast bounded method is almost 15 times as fast as the best NCE method and is still 4.3 times as fast as the fast NCE method. In all comparisons, the difference in obtained objective is small.

In the case of *yes_big*, the best NCE method obtains validation and test objectives of -6.27 and -6.28 with 85,223 seconds of computation. This is compared to validation and test objectives of -6.28 and -6.29 with 13,599 seconds of com-

putation for the best bounded method. The fast NCE method obtains -6.30 on validation and -6.31 on test with 25,282 seconds, while the fast bounded method obtains -6.30 on validation and -6.31 on test with 2,216 seconds. In this case, the differences in objective between the NCE method and the bounded method are even smaller, but the best bounded method is over 6 times faster than the best NCE method, while the fast bounded method is 38 times faster than the best NCE method and 11 times faster than the fast NCE method.

Overall, these results indicate that, despite its conceptual simplicity, our method far outperforms the NCE method in terms of convergence speed with only a negligible loss in model quality.

3.6 Conclusions

In this chapter, we presented an initial foray into applications of embedding models. We demonstrated the utility of our models for the case of sequence modeling, in particular for the application of playlist generation. By using a Markov assumption on the nature of the probability distribution over playlists and encoding the resulting transition distribution with a random walk in embedding space, we were able to formulate a sensible embedding model for the task. We verified the performance of this model by comparing it to baselines from the language modeling literature, and found a significant gain in performance over these models across data sets. The performance gain increases with the sparsity of the data set (in terms of the ratio of the number of observed bigrams to the vocabulary size). This is primarily due to the greatly increased performance of our models when predicting bigrams that were never observed

in training. We also showed that our bounded likelihood training method is effective in comparison to the naïve method using exact calculation of the partition function. Furthermore, the bounded likelihood method vastly outperforms the state-of-the-art NCE method in terms of convergence speed, while suffering only a negligible degradation in the quality of the resulting models.

Much of the work in this chapter was published previously at KDD 2012 [6]. One of the biggest modifications since then is the change in the training method. In the previous work, models were trained using naïve maximum likelihood training with exact calculation of the partition function. The efficient training method used now, described in Section 2.3, allows us to cope with significantly larger data sets.

However, the model described so far suffers from a problem which is common to many approaches to sequence modeling. Namely, although the model is well-suited to handle *bigrams* which were never seen in training, it still cannot cope with unobserved *songs* in its present form. This issue provides a good motivation for the work in the next chapter.

CHAPTER 4

INCORPORATING SIDE INFORMATION

The embedding-based playlist generation algorithm from the previous chapter is able to generate coherent playlists using only historical playlist data. This is achieved by embedding each of the songs in the training set into a metric space on which a probability distribution over playlist transitions is defined. However, two complementary motivations encourage us to dig deeper into the potential of this model. The first is a limitation of the model as defined so far, namely that songs which were never observed in training cannot be accommodated at test time. The second is the prospect of incorporating side information to form a richer model. In our particular application, we will consider adding social tag data in order to improve the model.

4.1 The Out-of-Vocabulary Problem in Playlist Generation

Recall that playlist generation is an application of the more general notion of sequence modeling, and that another popular application of sequence modeling is that of language modeling. In this latter case, we wish to model sequences of words similarly to the way we model playlists in the current work. A potential problem with sequence modeling in general is the *Out-of-Vocabulary* (OOV) problem, where words or objects may be encountered at test time that were never observed at training time.

According to Creutz et al [10], language modeling as it is applied to English does not suffer greatly from this problem. This is partly because English is a so-called *morphologically poor* language – that is, most words of similar meaning

are created from at most a couple of morphemes, or units of word meaning. Because of this, the vocabulary of English for the sake of language modeling is relatively well-defined, and the vast majority of observable words can be found easily in a decently large corpus of text. .

However, as the authors of [10] argue, other *morphologically rich* languages like Finnish or Turkish suffer greatly from the OOV problem. This is because words in these languages may be created from a potentially unbounded number of morphemes – many of them indicating instance-specific characteristics like syntactic structure, tense, person, possession, and more. The combinatorial nature of these morphemes creates a severe sparsity problem sequence models are applied to these languages on the basis of words. The authors of that work attempt to overcome this by instead applying sequence modeling at the morpheme level.

In the case of playlist generation, the OOV problem poses a serious challenge. Music streaming services collect massive music catalogs in order to attempt to satisfy their user base. In the specific case of Spotify, the music library consists of over 30 million songs, with over 20,000 added to the service each day[1]. Furthermore, in 2013 when the catalog consisted of about 20 million songs, the company reported that 20% of its available music had never been streamed[2]. This implies that the sparsity that gives rise to the OOV problem is indeed a concern in the realm of music information retrieval and for the task of playlist generation in particular.

[1] `https://press.spotify.com/us/information/`
[2] `https://news.spotify.com/us/2013/10/07/the-spotify-story-so-far/`

4.2 Side Information and Social Tag Data

Another challenge concerning playlist models is the question of how to effectively incorporate side information into the model. In addition to content-based approaches using audio features such as [23], a wealth of information about music is available on the Internet, such as text reviews of albums, news stories about artists, lyrics data, and social data. By incorporating some of this data directly into the model in a sensible way, we could make further progress in improving playlist generation methods.

In particular, in this work, we will develop a method for incorporating *social tag data* as mined from Last.fm. Last.fm is a music discovery service with a social component – users log their track plays, participate in forum discussions, and tag music with descriptive qualifiers. These tags are free form text that may describe any quality of the music – genre, emotion, instrumentation or musical features, or release dates are just a few examples of popular descriptors. Using a proprietary aggregation method, the service displays the most popular descriptions of individual tracks. By taking advantage of these crowd-sourced descriptors of the songs we model, we can develop a method that allows unseen songs to be modeled in the space using only these attributes. In this way, we can overcome the OOV problem for the task of playlist generation.

In addition to alleviating the OOV problem, our model also jointly embeds songs and tags in a semantic space. This yields a semantically meaningful distance metric that can be used to judge the similarity between tags and songs. We will demonstrate the benefits of this joint embedding for visual interpretability as well as for retrieval tasks.

4.3 Model Formulation

We incorporate the social tags into the model in a probabilistically sensible way by adding a prior on the position of each tagged song. In short, we assume through this prior that each song should be close in embedding space to the average embedded position of its tags. This way, model updates to the songs propagate to the tags corresponding to those songs, and the tags end up close in the embedding space to the songs that they describe – including those that were not explicitly labeled with the tag.

First, note that in this work, for the sake of simplicity and interpretability, we use a single point model for the songs. That is, there exists only a single embedding function $X(\cdot)$ for the songs, and we use the scoring function $s(s, s') = \Delta(X(s), X(s'))$. There also exists an embedding function $Y(\cdot)$ defined for each tag t in the tag vocabulary T and mapping to the same space in $\mathbb{R}^d$ as the song embedding X. With $T(\cdot)$ we denote a function that maps each song to the set of tags with which that song is labeled – a possibly empty set.

Now we formally define the prior which we apply to the tagged songs in the model. Specifically, we assume that the position of the song is distributed according to a Gaussian distribution with its center at the average tag position for this song:

$$P(X(s)|T(s)) = \mathcal{N}\left(\frac{1}{|T(s)|}\sum_{t\in T(s)} Y(t), \frac{1}{2\lambda}I_d\right) \tag{4.1}$$

This equation yields λ as a parameter of training, which can be interpreted as our prior belief about the strength of the assumption we place on the song posi-

tions. Larger values of λ lead to a low-variance distribution, forcing all tagged songs to lie very close to the means of their tags. Small values produce a weak assumption where songs are allowed to drift more freely. Note that we apply this prior only to those songs which are labeled with at least one tag in the data set – we do not make any assumptions about the positions of untagged songs. This allows us to continue to incorporate the knowledge about song to song relationships that we glean from the behavior of unlabeled songs.

Denote by $\bar{X}(s)$ the deviation of $X(s)$ from the tag mean of s – that is:

$$\bar{X}(s) = X(s) - \frac{1}{|T(s)|} \sum_{t \in T(s)} Y(t) \tag{4.2}$$

Then if we define $S_{tag} = \{s \in S \ : \ |T(s)| > 0\}$, the new bounded average log-likelihood with the prior incorporated takes the form:

$$LL_p(D) = LL_b(D) + \frac{\lambda}{|S_{tag}|} \sum_{s \in S_{tag}} \|\bar{X}(s)\|_2^2 \tag{4.3}$$

where $LL_b(D)$ is the bounded log-likelihood from Section 2.3. Now note that for all tagged songs s we have:

$$X(s) = \bar{X}(s) + \frac{1}{|T(s)|} \sum_{t \in T(s)} Y(t) \tag{4.4}$$

In our optimization, we interpret $\bar{X}(s)$ as a deviation or correction vector (which we store as a paramter vector in the model instead of the final song position $X(s)$) and use the chain rule to distribute updates to the tag vectors and correction vector. That is, we conduct training as in Algorithm 1 for a single point

model, except that whenever the vector $X(s)$ for a tagged song s receives an update u, we instead apply the following updates:

$$\begin{aligned} Y(t) &\leftarrow \frac{u}{|T(s)|} \; \forall \; t \in T(s) \\ \bar{X}(s) &\leftarrow u \end{aligned} \tag{4.5}$$

Now to penalize the deviation from the mean as defined in the prior, at each sampling iteration within the training algorithm, we sample a tagged song uniformly at random and perform the following update using the learning rate η chosen for the optimization:

$$\bar{X}(s) \leftarrow -\lambda\eta\bar{X}(s) \tag{4.6}$$

In this way, in expectation, these updates fulfill the role of optimizing the second term in Equation 4.3. Note that in this step as well, we use AdaGrad [12] in order to adapt the step sizes to each parameter given the history of the current optimization.

4.4 Experimental Evaluation

In our experiments, we aim to answer the following questions about our new model:

- What implications does the new model have for the interpreting and visualizing the model and data?
- How effective is the model in alleviating the OOV problem?

- Can the song/tag similarity from the resulting space be leveraged for retrieval tasks?

We will explore these questions in detail in this section. In all experiments, for the sake of interpretability and simplicity, we use the single point model. Unless otherwise indicated, all models are selected based on optimal performance on the validation set.

4.4.1 Data Set

We augmented the *yes_small*, *yes_big*, and *yes_complete*data sets from the previous chapter with social tags crawled from Last.fm. Namely, for each song, we queried Last.fm for the top tags for that song. Once we had crawled tags for all the songs, we retained the 250 tags that occurred most frequently among the songs in our data set. In the end, 80% of the songs in *yes_small* had tag information. In *yes_big*, the figure falls to 75%, and in *yes_complete* only 58% of the songs have tags. Including untagged songs, the average number of tags per song is 20.5 for *yes_small*, 15.2 for *yes_big*, and 7.8 for *yes_complete*. When the untagged songs are excluded, the averages become 25.0, 20.2, and 13.4 for *yes_small*, *yes_big*, and *yes_complete*respectively.

4.4.2 A View of Tag Embeddings

In Figures 4.1, 4.2, and 4.3, we see visualizations of the joint song and tag space in two dimensions, along with labels for many of the most frequently occurring tags. These figures again show the clear genre segmentation of the model space

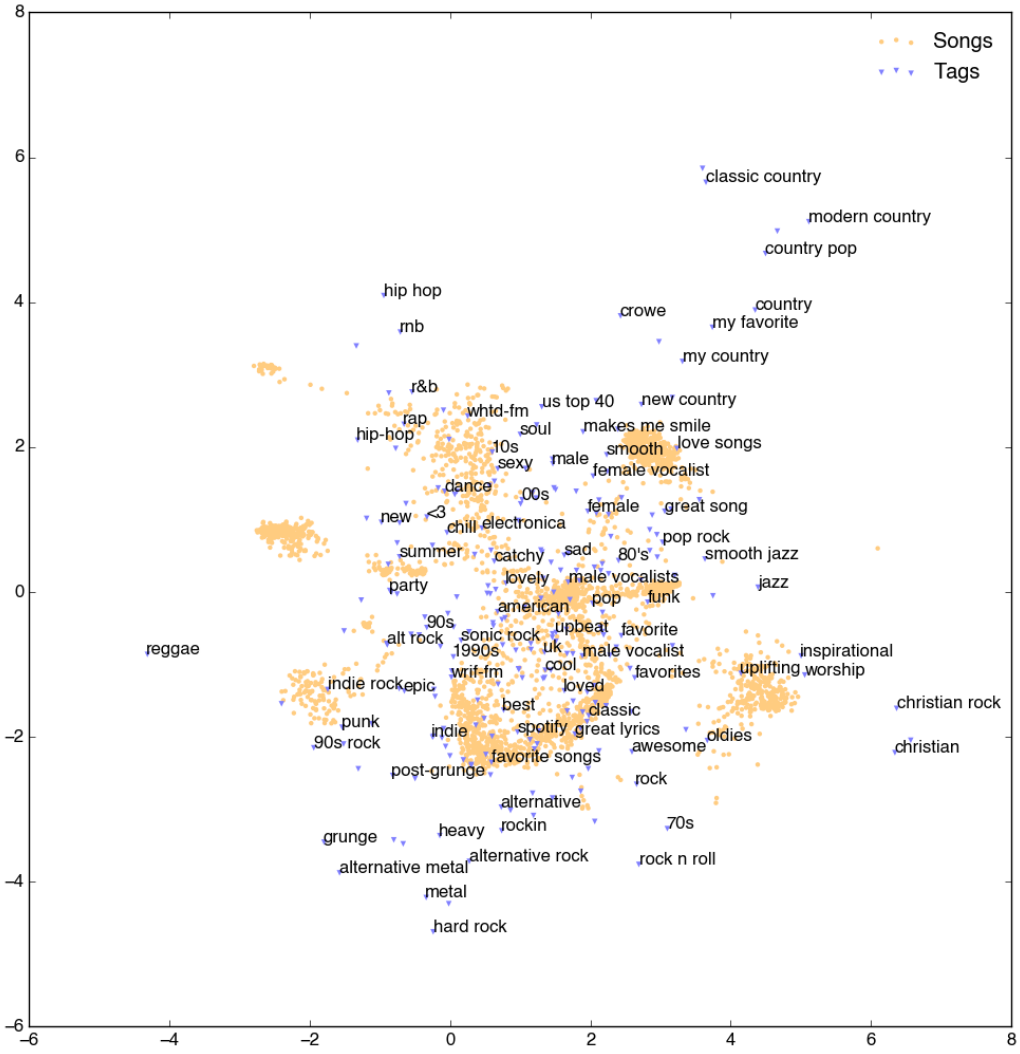

Figure 4.1: Visualization of a two-dimensional embedding model trained on *yes_small* with tags.

that was seen in the previous chapter. However, with the tags, we can now explicitly see the genres and musical characteristics defined by the different regions of the space. Furthermore, we can see how tight and clearly defined some of these groupings can be: for example, the tags *r&b*, *rnb*, *hip-hop*, *hip hop*, and *urban* repeatedly group closely together, as do the tags *hard rock*, *metal*, *alternative metal*, *grunge*, and *alternative rock*. These groupings reveal another confirmation

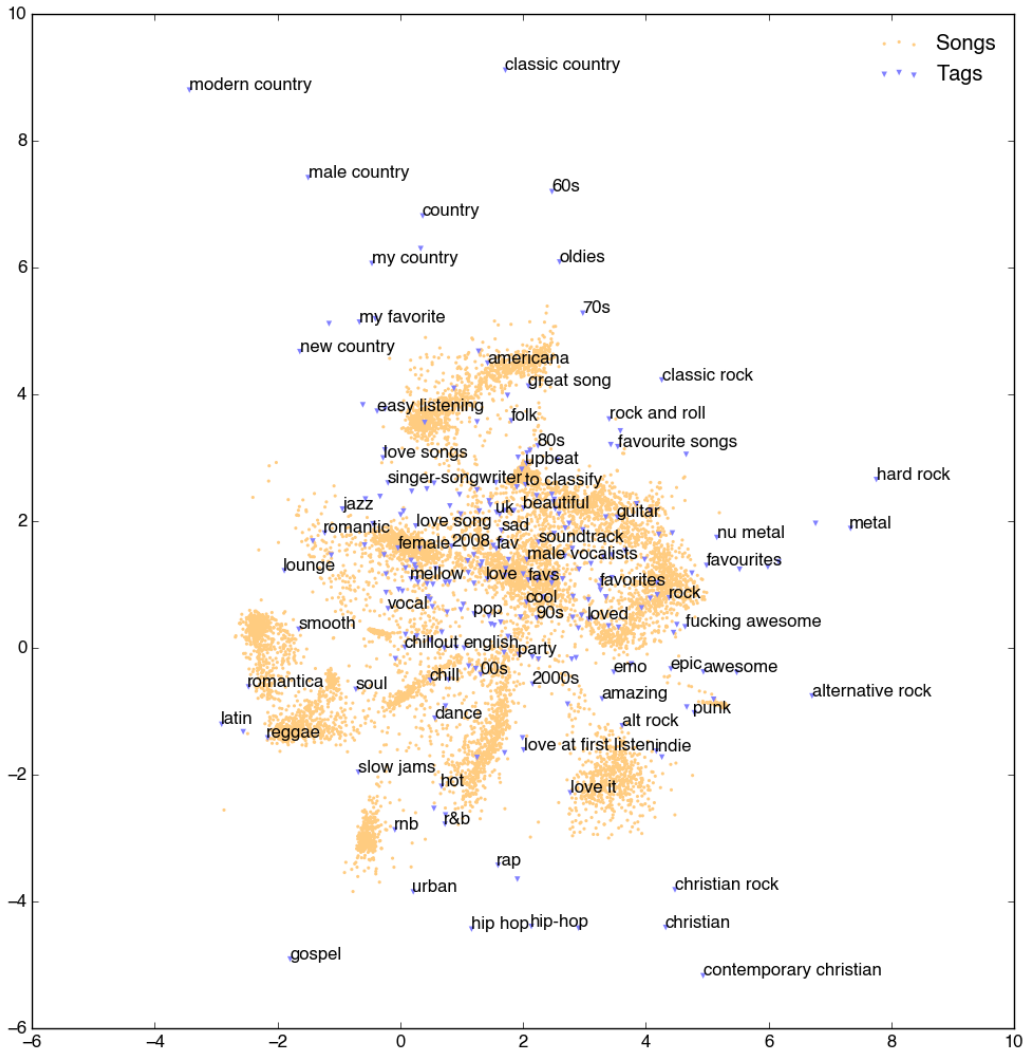

Figure 4.2: Visualization of a two-dimensional embedding model trained on *yes_big* with tags.

of the semantic meaning of the space: multiple spellings of the same descriptor that arise from the free-form crowdsourced labels, such as *r&b* versus *rnb* and *hip-hop* versus *hip hop*, always appear near each other in the space.

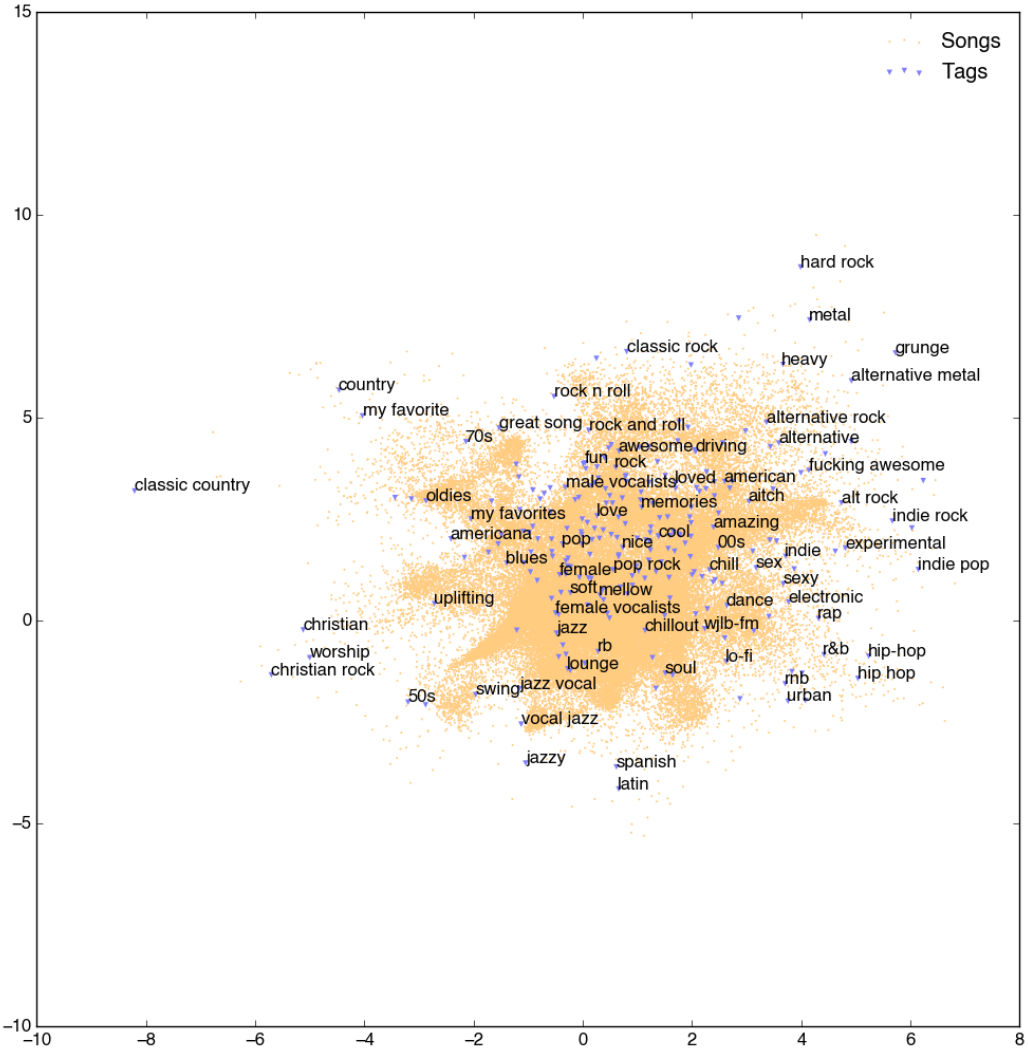

Figure 4.3: Visualization of a two-dimensional embedding model trained on *yes_complete* with tags.

4.4.3 Out-of-Vocabulary Performance

As mentioned previously, a primary motivation for developing the tag embedding model was to alleviate the OOV problem – the challenge of modeling songs that were never seen in training. Fortunately, the prior we employ to incorporate the tags makes it simple to incorporate these unseen songs as long as at

least one tag is provided for the song. This is accomplished by considering that the most likely position for a song s with tag set $T(s)$, according to our prior, is:

$$X(s) = \frac{1}{|T(s)|} \sum_{t \in T(s)} Y(t) \tag{4.7}$$

Thus, the projection of an observed song into the model is performed by simply averaging the positions of the song's tags.

In order to verify the performance of the tag model on the OOV modeling task, we design an experiment simulating the conditions of predicting OOV songs. Specifically, we pool all the available playlist data for a data set and choose one tenth of the songs to place into a validation set and one tenth of the songs for the test set. Then, we create a training set of song bigrams by including only bigrams where both songs are in the training set. The validation bigram set consists of bigrams where either one song is in the training song set and the other is in the validation song set or both songs are in the validation song set. The test bigram set is constructed similarly to the validation bigram set. We then choose the optimal model in terms of performance on the validation set and finally report the performance of this model on the test set. This validation is performed over each λ in $\{0.0, 0.001, 0.01, 0.1, 0.5\}$ and each η in $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$.

In Figures 4.4 and 4.5, we show the results of the OOV modeling task. In these figures, we compare only against the uniform baseline, since neither the unigram nor the bigram distributions are applicable to the task. We can see that our model vastly outperforms the baseline at all dimensions on both data sets, even though there are many bigrams which only contain unseen songs. Additionally, the optimal values of λ on *yes_small* were 0.1 for dimension 2 and 0.01 for every other dimension, and the optimal values for *yes_big* were 0.001,

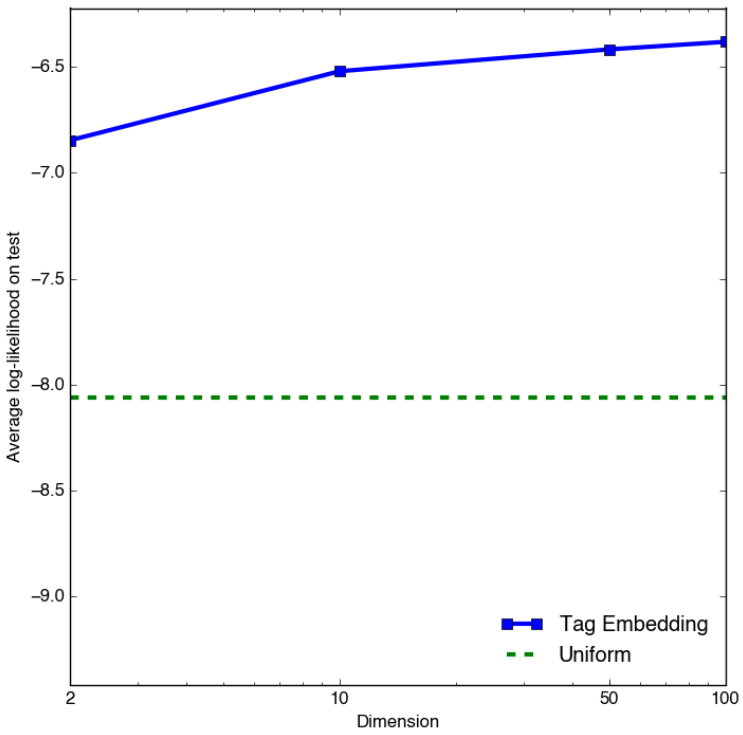

Figure 4.4: Results on the OOV modeling task on the *yes_small* data set.

0.001, 0.0, and 0.01 for dimensions 2, 10, 50, and 100, respectively. This implies that only a light penalty is required for the drift of a song from its tag mean.

4.4.4 Performance as a Function of Prior Variance

The prior we have incorporated into the model allows us to accurately model OOV songs, but the parameter λ must be set appropriately in order to perform optimally on that task. For sufficientlly small settings of λ, the song embedding positions $X(s)$ become unconstrained, and the tag model can learn the same embedding as the model without tags, yielding the same performance as the untagged model. However, if λ is large enough, the constraint that the song

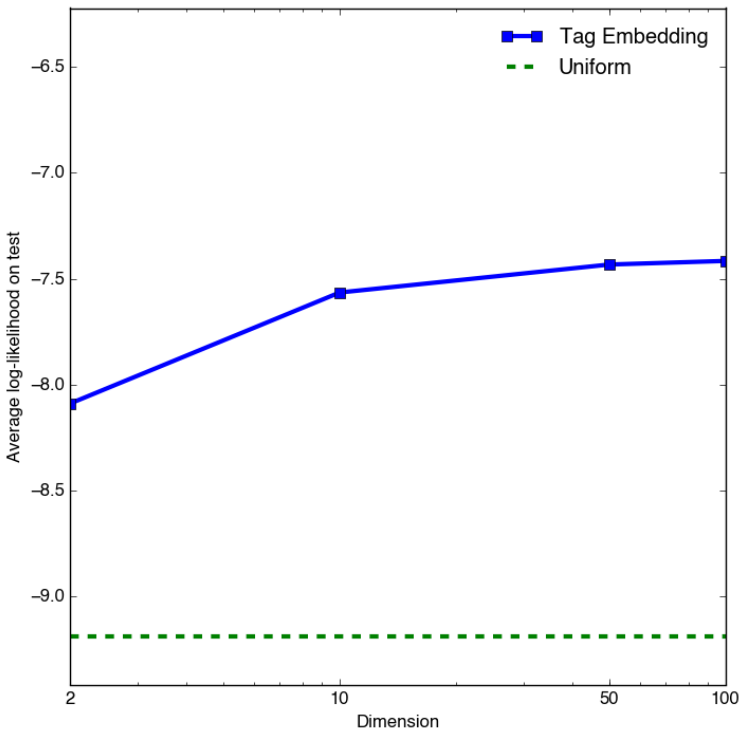

Figure 4.5: Results on the OOV modeling task on the *yes_big* data set.

positions lie close to their tag means may harm the performance of the playlist model. Therefore, it is important to consider the effect of this parameter on the playlist model we obtain.

We show the effect of this parameter in Figures 4.6 and 4.7 on the *yes_small* and *yes_big* data sets, respectively. In these experiments, we used the train/validation/test split from Chapter 3, where the split is performed on the basis of playlists. We trained models of various dimensions and settings of λ, validating over the step size parameter η. Test performance is shown for the models that performed best on the validation set. In addition, the performance of the best untagged single point model (which is independent of λ) is shown in each case for dimensions 2 and 100.

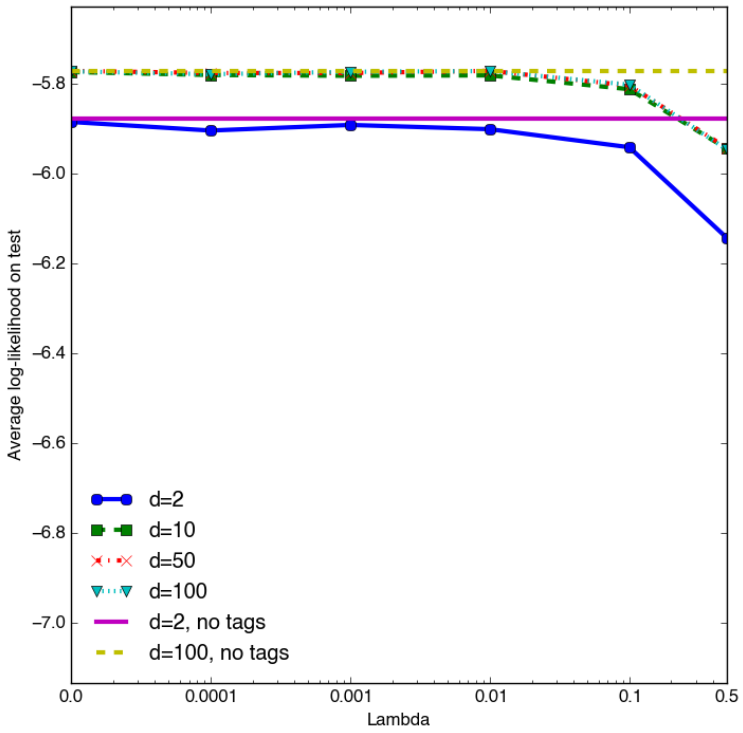

Figure 4.6: Tag model performance on *yes_small* plotted against λ, the strength of the prior assumption. The performance of the optimal untagged model for dimensions 2 and 100 is also plotted.

We can see from these plots that performance is stable and comprable to the untagged model until λ reaches 0.1, a fairly large value. At 0.1, performance degrades slightly, but is still barely worse than the untagged model. The largest value considered, 0.5, yields a more significant decrease in test log-likelihood, but even this difference may be acceptable if a large value is needed to ensure good OOV performance. However, it is important to consider these figures in the context of the previous experiment. Namely, the optimal value for OOV performance in the previous subsection was never larger than 0.1, which was optimal only for dimension 2 on *yes_small*. For other dimensions, *yes_small* only required $\lambda = 0.01$, and *yes_big* did not require a value larger than 0.01 for any dimension. This suggests that the optimal value for overcoming the OOV prob-

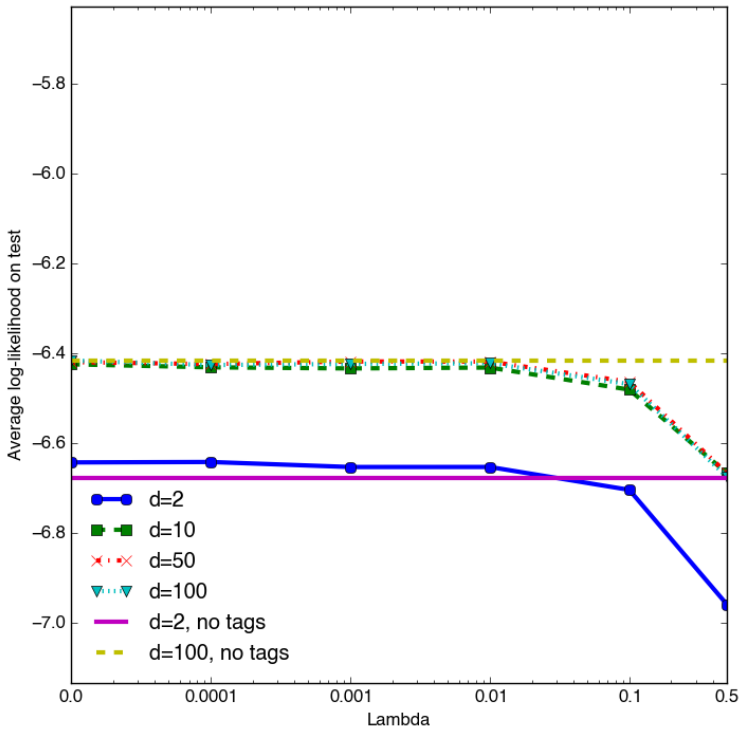

Figure 4.7: Tag model performance on *yes_big* plotted against λ, the strength of the prior assumption. The performance of the optimal untagged model for dimensions 2 and 100 is also plotted.

lem still results in models that are just as good for playlist generation as any untagged model.

It is also interesting to note that for *yes_big*, the best tagged models actually outperform the best untagged model. This could be due to two different factors. First, by forcing the model to rely on the tag vectors to represent each song, the model gains transfer from the shared tag parameters between songs. Namely, songs that occur rarely in the training data must nonetheless lie close to their tags in the space, and if these tags are shared by similar songs with many observations in the data, then the rare song is encouraged to sit near its "true" semantic position in the space. Second, the optimization problem is non-

concave, so there could be many local optima in which the model could get trapped. However, the dependence of the songs' parameters on the tag parameters might change the potential trajectory of the optimization and help avoid or overcome local optima.

4.4.5 Song-Tag Similarity and Retrieval

Jointly embedding songs and tags into the same space gives us a natural similarity metric between the two – closeness in Euclidean space. In order to demonstrate how meaningful the space is in this regard, we develop a retrieval task that leverages this notion of similarity. The task is a query-by-tag song retrieval task, wherein a tag is selected (e.g. by a user) for a query, and the goal is to return a ranked list of songs that are relevant to that tag.

In terms of experimental design, for each data set, we split the tagged songs randomly into train, validation, and test sets which comprise 80%, 10%, and 10% of the tagged songs respectively. In training, we train on the full set of song bigrams, but with the tags for the validation and test sets hidden from the model. At test time, we query each tag present in the validation/test set and use proximity in the embedding space to rank the songs found in the validation/test set. Any song which has the query tag in its original labels is considered a relevant result. In this way, we check the performance of the model on the retrieval task when no explicit tag information is known about a song whatsoever. The notion of song/tag similarity for these songs is defined entirely through transfer – the songs lie close to similar songs in the space, and the tagged songs among these will lie close to their tags.

Since this is a ranking task, we use two ranking performance metrics to check the performance of the model. The first is the *precision at 10*, which is the proportion of the top 10 results which are relevant. The second is the *area under the ROC curve*, or AUC, which is a measure that balances the true positive rate and false positive rate of the top k results for all k. We also define two baselines for comparison, a random ranking and a frequency-based ranking. In the random ranking, the songs in the test set are ranked in a totally random order. In the frequency-based ranking, the songs are ranked in order of observed play count in the training set.

We divided the tags into several categories in order to get more insight into the strengths and weaknesses of the model for the task:

- **Genre:** this group consists only of genre descriptors like *rock*, *hip hop*, and *reggae*. This group contained 83 tags in *yes_small*, 86 tags in *yes_big*, and 89 tags in *yes_complete*.
- **Emotion:** these tags include descriptions of emotions. *yes_small* has 37 tags in this group, while *yes_big* and *yes_complete* both have 39.
- **Musical:** this refers to tags which describe technical aspects of a song like *minor key tonality*, instrumentation like *guitar* or *male vocals*, and similar non-genre musical aspects. There are 26 tags in this group in *yes_small*, 22 in *yes_big*, and 23 in *yes_complete*.
- **Time:** these refer to decades and years. There are 17 time-related tags in *yes_small* and *yes_complete*, and 15 in *yes_big*.
- **Other:** This is a catch-all group for the tags that do not fit into the other categories. Many of these praise the song (*best song ever*, *my favorite*, *good*

song), and some others are geographical in nature (either explicitly as in *california*, *uk*, *usa* or implicitly as in radio station call letters like *wrif-fm* and *wkqi-fm*). This group contains 87 tags in *yes_small*, 88 in *yes_big*, and 82 in *yes_complete*.

For our model, we considered models trained for all λ in $\{0.0, 0.001, 0.01, 0.1, 0.5\}$ and each η in $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$. We restricted dimensionality to 100, since this higher dimensionality captures more of the signal of the data and performed best on each data set. Note that each result offers a worst-case scenario analysis. First, we consider only songs that had no tag information available in training. Second, many tags are semantically the same despite slightly different spellings (recall *hip-hop* and *hip hop*), but we only count a result song as relevant when it has exactly the same tag as the query tag.

In Figures 4.8 and 4.9 we see the results of the retrieval task. As we expect, in each case the random baseline achieves an AUC of close to 0.5 and a precision at 10 of about 0.1. The frequency baseline offers slight improvements over this in AUC, but is actually worse than random in terms of precision, implying that for many queries, the most popular songs are not very relevant. It is also interesting to note that for both metrics and both data sets, the frequency baseline improves slightly on the time category. This could be due to the fact that, despite the presence of a few tags for somewhat older songs, there is a heavy bias towards more recent time periods in the tags. For example, in *yes_small* 8 of the 17 time tags refer to a time period from the year 2000 onward (the earliesst being the generic *2000s* and *00s* with the next earliest being *2008*), 3 refer to a time period in the nineties (*90's*, *1990s*, and *90s*), 3 to the eighties, 2 to the seventies, and only 1 to the sixties. Especially considering the source of the playlists – radio

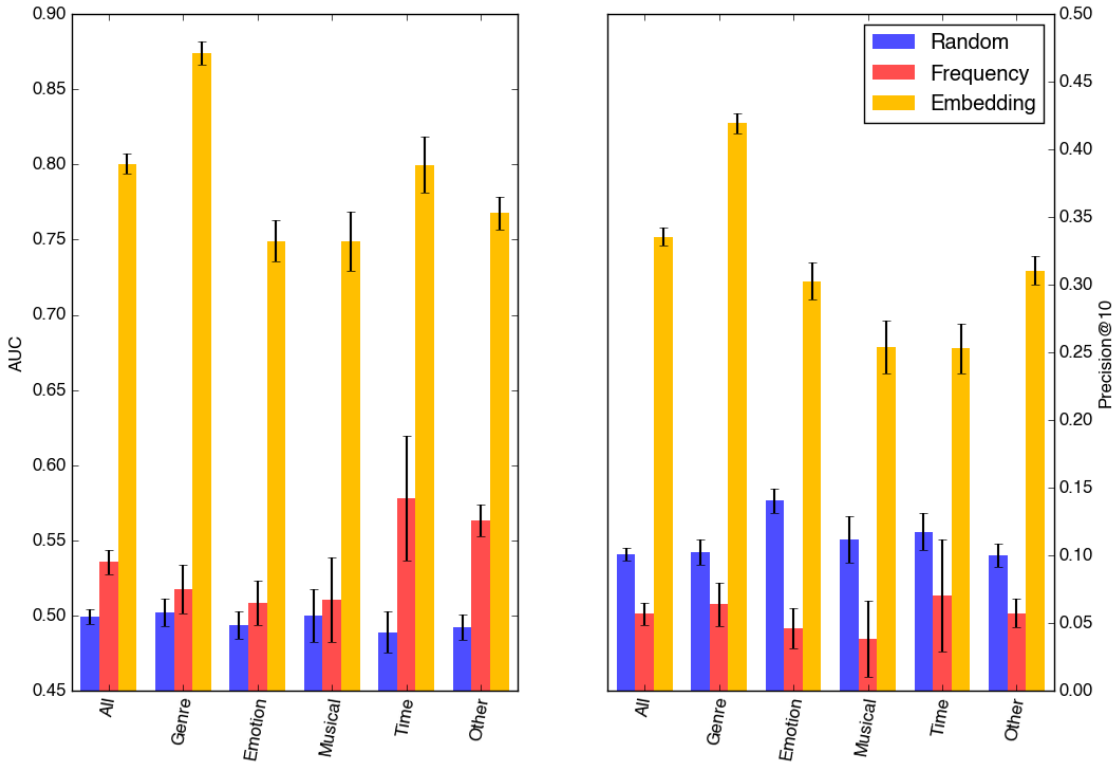

Figure 4.8: Results on query-by-tag retrieval task on the *yes_small* data set.

stations, many of which are probably commercial – it would be reasonable to expect that a number of these would be biased towards very recent music. Thus, the high play count songs are more relevant on average for most of the tags in this category.

In every case, our embedding method vastly outclasses the baselines, with an overall AUC of about 0.8 and an overall precision of about 0.35 in both cases. When the task is restricted to genre tags only, the performance becomes stronger – AUC increases to almost 0.87 and precision increases to around 0.46. This implies that on our playlist data sets in general, our models are best at capturing the semantics of genre. In both cases, performance seems to be worst on the musical tags. While it is imaginable that some of these correlate with genre to

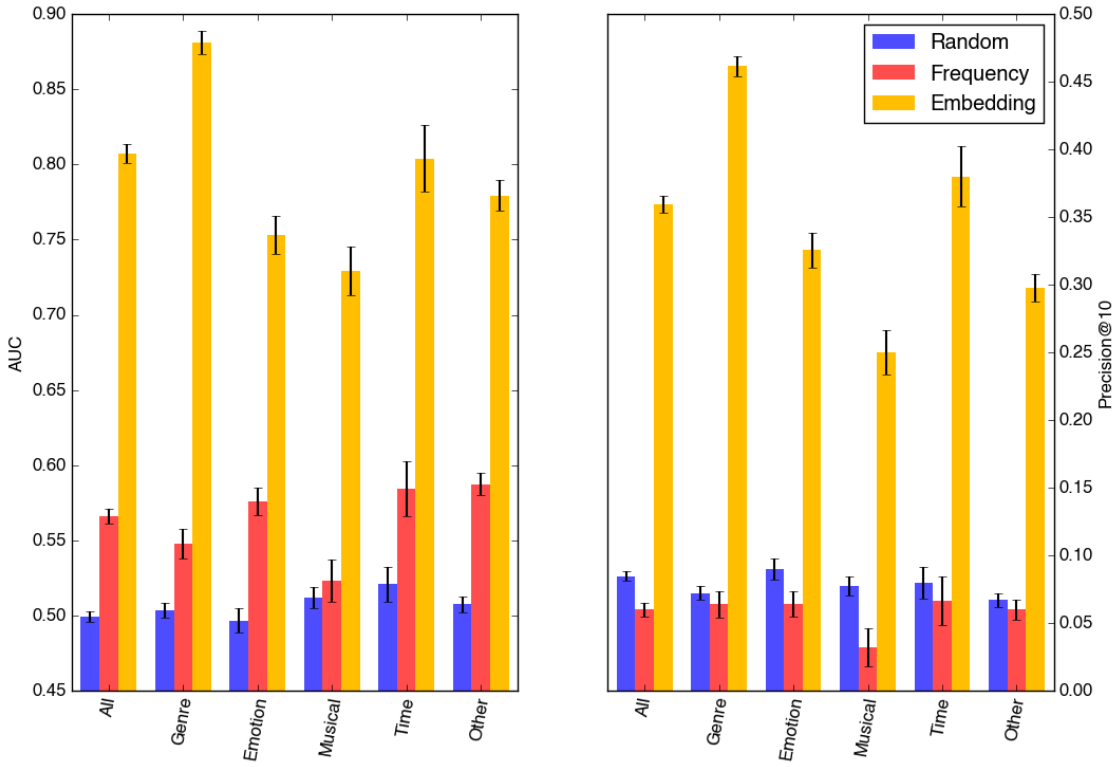

Figure 4.9: Results on query-by-tag retrieval task on the *yes_big* data set.

some extent (such as *guitar hero*, *guitar virtuoso*, *guitar solo*, and *awesome guitar jams*), many of the tags are more generic (*instrumental*, *male vocalist*, *female vocalists*) and are not as semantically segmented in this space trained for largely genre-dependent playlisting.

4.5 Conclusions

In this chapter, we presented a method for incorporating side information (in the form of social tags) into an embedding model in order to overcome the OOV problem. Our new model takes advantage of the tags through the use of a prior on the positions of tagged songs. Since the tags are added to the model in this

fashion, the new model remains well-founded probabilistically. As a result we have a playlist generation model which can use tags for interpretability and visualization and for projecting unseen songs into the model in a simple and sensible way. The joint song/tag space also presents a similarity metric which can be used to judge a song's characteristics, especially in terms of genre. This similarity metric can be used effectively as part of a query-by-tag song retrieval system.

The ideas in this chapter were previously explored by the author and colleagues in [32]. Since that work was conducted, we have developed more efficient training methods, which allowed us to more deeply explore the data we collected and train models on larger data sets.

In this chapter as well as Chapter 3, we have noted the expressive power of visualizations of two-dimensional embedding models. These models allow us to quickly glean interesting patterns from large, unstructured collections of data, such as historical playlist data. In the next chapter of the work, we will shift away from modeling strictly for the sake of generative tasks like playlisting. Instead, we will begin to consider the embedding model as a tool for data analysis. In other words, the modeling task at hand has so far been the end and the embedding itself was simply a means to this end. In the next chapter, we will view this differently – the embedding and its visual and spatial properties will be our end, and we will develop the modeling task largely in order to achieve the goal of an interpretable space which easily displays its structure and patterns.

CHAPTER 5

TEMPORAL MODELING OF MUSIC LISTENING HISTORIES

In Chapters 3 and 4, we developed embedding models to tackle the problem of playlist generation. In the process, we saw that these embedding models are highly interpretable, and that visualizations of these models enable us to easily find patterns present in the data. The spaces tend to have rich semantic meaning, and relationships among the various objects can be clearly observed via their proximity. In this chapter, we shift towards leveraging this visual interpretability for the sake of data analysis. The models prove to be a powerful tool for finding patterns in the data. We will begin with a temporal analysis of a long-term listening data set.

5.1 Long-Term Music Listening Data

Of the numerous sources of music listening data available on the web, one of the most intriguing is the case of Last.fm. Last.fm provides a wealth of information about music – tag information as seen in the previous chapter, play counts for artists, show listings, and more. In this chapter, we will focus on the case of user listening histories.

One capability that Last.fm provides to its users is the ability to install an application known as a *Scrobbler*. The Scrobbler records the user's track plays and reports them to Last.fm, where they are logged and displayed on the user's profile. Users can then view their track history as well as top artists for the week, month, year, and for all time. The track logging is somewhat social in context – users can not only view their own track history, they can also view other users'

track histories and compare their taste compatibility.

Notably, the entire track history for each user can be viewed, extending all the way back to the first track that a user logged after created a profile. This means that the track histories for a large enough pool of users, in aggregate, offer a record of overall tastes and trends in the collective music histories of the users. However, the data is largely unstructured and discrete – we simply have a collection of users, each with a collection of tracks with timestamps. Therefore, in order to analyze the data, we develop an extension of our embedding models that incorporates users and songs, but which also adds temporal dynamics to the model in order to capture the evolution of music listening behavior over time.

5.2 Modeling Users, Songs, and Time Dynamics

5.2.1 User-Sensitive Playlist Model

First, we define the user-sensitive playlist model, without time dynamics, which will form the basis of our temporal models. This model is similar to the playlist models we have employed in previous chapters, but with an added elemnt of context for the user who is currently listening to the playlist. Specifically, we consider an embedding task where our context set C consists of user/song pairs (u, s_{cu}) and target objects s_{ne}. In this case, u represents the currently listening user, s_{cu} represents the currently playing song, and s_{ne} represents the candidate song for the next track play. Our goal is to learn a set of conditional probability distributions $P(s_{ne}|u, s_{ne})$ for each of these contexts.

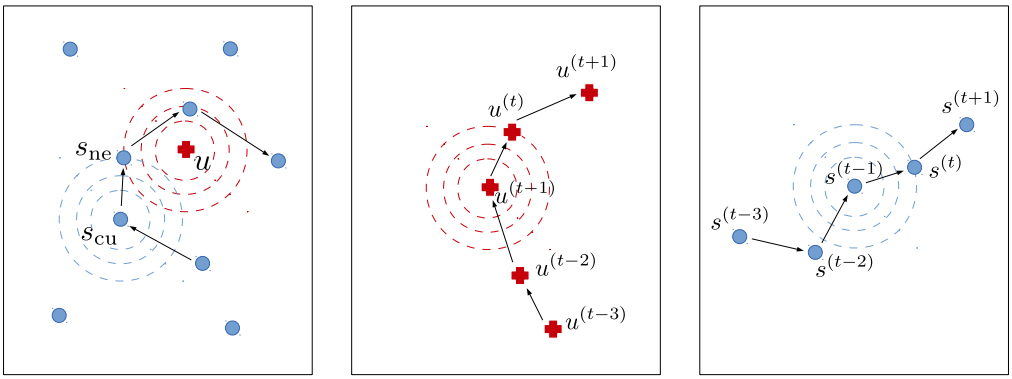

Figure 5.1: Illustrations of the embedding models. Blue dots and red crosses represent songs and users respectively. On the left, a static playlist model. A playlist is represented by songs that are linked by arrows. The next song s_{ne} is decided by both current song s_{cu} and the user u (The bias term also has its effect, which is not shown here). In the middle, the drifting of a user u over timesteps in the user-dynamic model. At each timestep, a random walk governed by a Gaussian distribution is taken. On the right, a similar drifting of a song s over timesteps in the song-dynamic model.

Following the terminology of Chapter 2, we will accomplish this by learning two embedding functions: $X(\cdot)$ for the users, and $Y(\cdot)$ for the songs, and in Equation 2.5 we set $X_1 = X$ and $Y_1 = Y_2 = Y$ to obtain the scoring function:

$$s((u, s_{cu}), s_{ne}) = -\Delta(X(u), Y(s_{ne})) - \Delta(Y(s_{cu}), Y(s_{ne})) + b_{\text{idx}(ne)} \tag{5.1}$$

This translates to the following conditional probability:

$$P(s_{ne}|u, s_{cu}) = \frac{\exp(-\Delta(X(u), Y(s_{ne}))^2 - \Delta(Y(s_{ne}), Y(s_{cu}))^2 + b_{\text{idx}(s_{ne})})}{Z(s_{cu}, u)}. \tag{5.2}$$

In this way, we consider a candidate song s_{ne} to be a likely candidate for a track play if its representation $Y(s_{ne})$ is close to both the listening user's repre-

sentation $X(u)$ and the currently playing song's representation $Y(s_{cu})$. The term $b_{\text{idx}(ne)}$ indicates a bias term for the candidate song – a song is also a better candidate if it is a popular song in the data set. The distance relations of this model are illustrated in the left panel of Figure 5.1.

In the learned embedding model, we obtain three measures of similarity. First, proximity between a user and a song reflects a high propensity of that user to play that song – that is, the song is a good fit for that user's taste profile. Second, proximity between two songs implies similarity between those two songs – similar users will appreciate the song, and the two will work well next to each other in a playlist. Third, users that are near each other in the space should have similar taste profiles.

5.2.2 Modeling Temporal Dynamics

We model changes in user preferences as a stochastic process on a macroscopic level. In the following experiments, each macroscopic timestep $t \in \mathcal{T}$ ($\mathcal{T}$ is the set of all timesteps) denotes a quarter of a year, and notation like 20083 denotes "third quarter of year 2008".

Let us first consider a macroscopic stochastic process where positions of users are changing over time, while the position of the songs are fixed in the latent space. Denoting with $u^{(t)}$ the position of user u in embedding space at timestep t, the overall trajectory of a user is $u^{(*)} = (u^{(1)}, u^{(2)}, ...)$. At each timestep t, the microscopic transition probability $P(s_{ne}|u^{(t)}, s_{cu})$ now depends on the users current position, and the conditional probability of the next song is

$$P(s_{ne}|u^{(t)}, s_{cu}) = \frac{\exp(-\Delta(X(u^{(t)}), Y(s_{ne}))^2 - \Delta(Y(s_{ne}), Y(s_{cu}))^2 + b^{(t)}_{\text{idx}(s_{ne})})}{Z(s_{cu}, u^{(t)})}. \tag{5.3}$$

Note that even though the positions of songs are fixed, we still give each song a time-varying popularity term $b_i^{(t)}$.

To restrict users from drifting too much from one timestep to the other, we model a users trajectory as a Gaussian random walk. The middle panel of Figure 5.1 illustrates such a random walk. Concretely, this means that the user's next position $u^{(t)}$ is a Gaussian step $\mathcal{N}(u^{(t-1)}, \frac{1}{2\nu_{\text{user}}} I_d)$ from the current position $u^{(t-1)}$. Here, I_d is the d-dimensional identity matrix, and ν_{user} is inversely related to the variance (which can be viewed as a regularization coefficient that influences step sizes). This Gaussian distribution makes it more likely that the user's positions at two consecutive timesteps are close to each other.

Considering both the stochastic process over transition triples and the stochastic process describing the users' trajectories, the overall user-dynamic embedding model can be trained via maximum likelihood. If we set $\mathcal{T}_{-1} = \{t \mid (t \in \mathcal{T}) \wedge (t-1 \in \mathcal{T})\}$, then the average log-likelihood objective is:

$$LL_{ud}(D) = LL_b(D) - \nu_{\text{user}} \sum_{u \in U} \sum_{t \in \mathcal{T}_{-1}} \Delta(X(u^{(t-1)}), X(u^{(t)})) \tag{5.4}$$

where the song and time-dependent user positions are optimized to maximize the likelihood of the observed playlists.

5.2.3 Song-Dynamic Embedding Model

Similar to the user-dynamic embedding model, we also consider a song-dynamic embedding model which fixes the position of users and allows songs

to drift over time. In this model, the probability of each transition triple is

$$P(s_{ne}^{(t)}|u, s_{cu}^{(t)}) = \frac{\exp(-\Delta(X(u), Y(s_{ne}^{(t)}))^2 - \Delta(Y(s_{ne}^{(t)}), Y(s_{cu}^{(t)}))^2 + b_{\mathrm{idx}(s_{ne})}^{(t)})}{Z(s_{cu}^{(t)}, u)}. \tag{5.5}$$

After introducing an analogous Gaussian random walk for songs over different timesteps (as illustrated in Panel (c) of Figure 5.1), we get the average log-likelihood

$$LL_{sd}(D) = LL_b(D) - \nu_{\mathrm{song}} \sum_{s \in S} \sum_{t \in \mathcal{T}_{-1}} \Delta(Y(s^{(t-1)}), Y(s^{(t)})) \tag{5.6}$$

where users and time-dependent song positions are optimized.

From a technical perspective, it is conceivable to train an embedding model with both users and songs varying their position over time. We briefly explored this model, but found it difficult to interpret the resulting trajectories. We therefore focus on the restricted models in our empirical evaluation.

5.3 Experiments

Our experiments revolve around a *Last.fm* data set which we crawled using the site's API[1]. The crawl was conducted over the course of several weeks in the fourth quarter of 2012. Although it is unused in this work, we were initially also interested in the social network data, so we crawled through the social network using the top listener for each of the 10 top artists on the site at the time as seeds. For each user, we crawled the user's complete timestamped track history and friends list. We later augmented this data with the age, gender, and country of

[1] http://www.last.fm/api

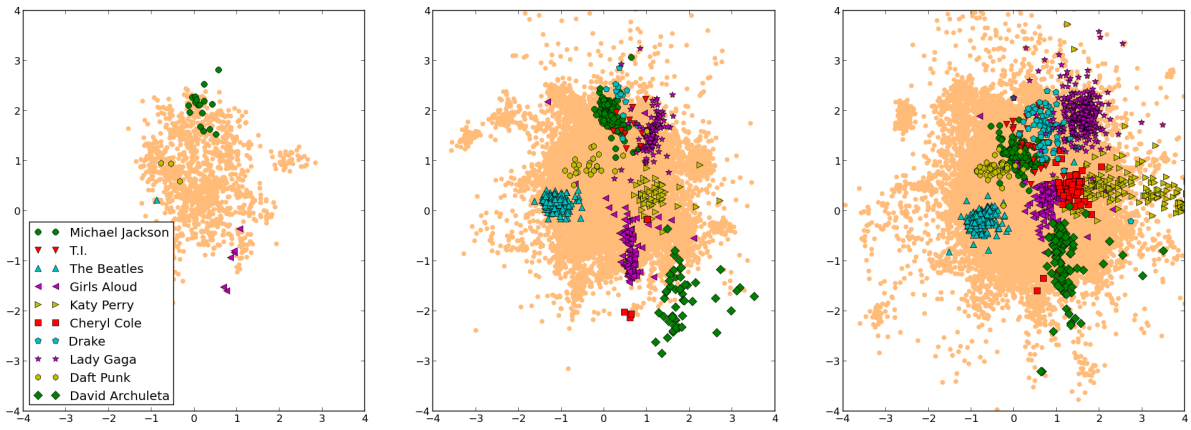

Figure 5.2: The song-dynamic model's song space plotted (from left to right) at 20051, 20091, and 20124

each user (for those for which it was available). We also crawled the tags for some of the songs, although we do not take advantage of this data in this work.

The result contains over 300,000,000 track plays by roughly 4700 users, with over 550,000 unique tracks. This data contains many noisy track names, so we pruned the data further by only considering tracks with at least 1000 plays and discarding users with no remaining track history after infrequent songs are discarded. This yields the set of track histories used in the experiments, which contains 4,551 users, 32,401 unique tracks, and roughly 200,000,000 track plays. We used this to create our "per-user playlist" data by splitting the track histories into playlists of consecutive songs that were played within 15 minutes of each other. Finally, we quantized the timestamps to divide each user's track history into year quarters, ranging from first quarter, 2005 until fourth quarter, 2012, for a total of 32 timesteps. From this point on, we will refer to the nth quarter of year *yyyy* as *yyyyn*, such as 20051 for 2005 first quarter.

We considered models with 2 dimensions in this work for the sake of sim-

plicity and ease of visualization. In order to find good values for ν_{song} and ν_{user}, we further divided the data by placing each fifth song transition into a validation set and the rest into the training set. We then used these to validate for the optimal values of these parameters. The user-dynamic model performed best with a low value of ν_{user}, with its optimal value at 0.01. In contrast, the song-dynamic model performed best with strong regularization, and the optimal ν_{song} was found to be 2.0.

5.3.1 Demographics of Users

The demographics of the data set reflect characteristics of the average Last.fm user. For each demographic category, we report percentages based on the number of users reporting in that category. 83% reported an age, 89% reported a country, and 91% reported gender. In our data, about 78% of the users are male, and about 88% are between the ages of 15 and 25 (roughly evenly split between the two groups) as of the crawl in 20124. The median user age is 20, and the average is about 20.8. Due to the social network crawl and a coincidence of the seed users, roughly 57% of our users are from Brazil. The country distribution has a fairly long tail, with only 84% coming from the 10 most popular countries, and 91% coming from the 20 most popular countries. The ten most well-represented countries in the data set are Brazil (57%), US (8%), UK (4%), Poland (3%), Russia (2.6%), Germany (2.3%), Spain (1.6%), Mexico (1.6%), Chile (1.3%), and Turkey (1.1%).

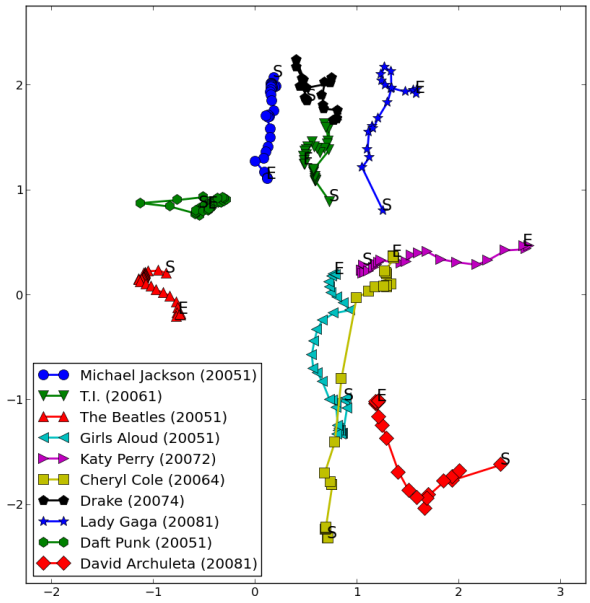

Figure 5.3: Artist trajectories over time. The legend gives the first quarter in which each artist was observed

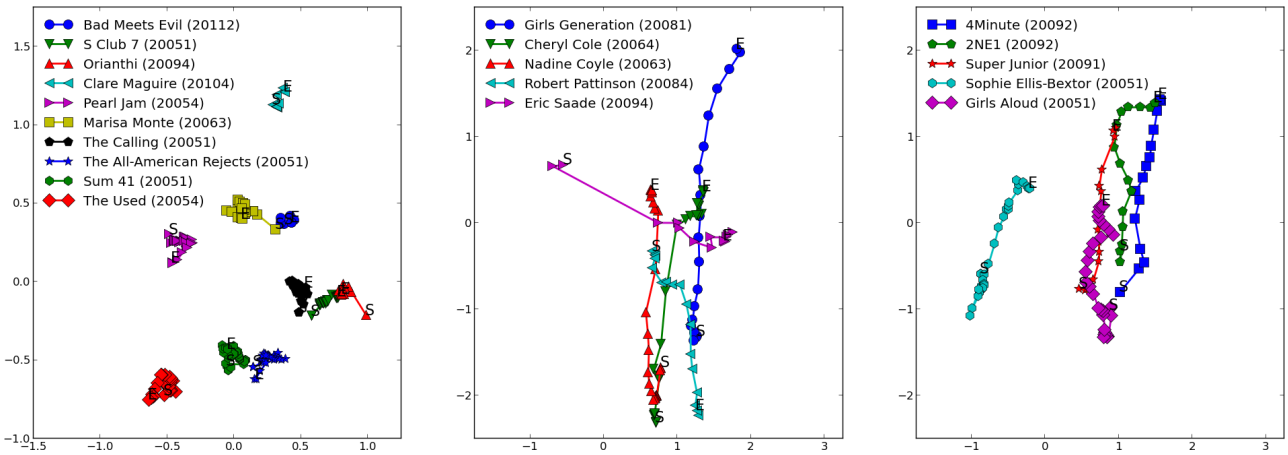

Figure 5.4: The 10 artists with the smallest variance in position over time (left) and the 10 with the largest variance in position over time (top 5 in center, next five at right). The first timestep at which each artist was observed is listed in parentheses.

5.3.2 Song-Dynamic Model

In the song-dynamic model, songs can move over time through a map of users. Among other things, the resulting trajectories give insight into how the appeal of songs and artists changed over time.

In Figure 5.2, we show the embedding of the songs at the start, middle, and end of the time sequence (i.e., timesteps 20051, 20091, and 20124). A song is plotted once it has been played at least once, which explains why the space becomes more dense over time. The locations of users are not plotted to reduce clutter. Generally speaking, the density of users is greatest around the origin and then decreases outwards. In this sense more popular music lies in the center, but note that we also capture popularity through the specific song bias parameter.

Are similar songs embedded at similar locations? To illustrate the semantic layout of the embedding space, we highlight the songs of some reference artists. Note that the songs of the reference artist cluster even though our embedding method has no direct information about artists. This verifies that the model can indeed learn about songs similarity merely from the listening patterns. We also note that our intuitive notion of artist similarity generally matches the distance at which our model positions them in embedding space.

How do songs and artists move? Figure 5.2 also shows that the songs of some artists move in the embedding space, while others remain more stationary. The artists' changes are aggregated into trajectories and displayed in Figure 5.3. Each dot in Figure 5.3 indicates the mean location of the songs of one artist at a specific time step. This plot enables us to see more clearly some events and trends in the music world that influence the model.

First, note that Michael Jackson's trajectory starts off clumped together in the same space, moving very little. Then, after some number of timesteps, it starts moving quickly towards the center. Upon closer inspection, the turning point in this trajectory turns out to line up exactly with the death of Michael Jackson in June, 2009.

Similarly, the Beatles start to drift slightly away from the center as many other artists enter the model. Then, they make an abrupt turn back towards the center. This aligns with the release of the Beatles' full catalog on iTunes in the 20104 after being totally unavailable via digital distribution before then.

Daft Punk also starts to drift away from the center until the release in December, 2010 of the motion picture *Tron: Legacy*, which featured a popular soundtrack by the duo.

We can also see Girls Aloud and Cheryl Cole (of Girls Aloud) drift from the edges rapidly towards the center in correlated paths, and the emergence of David Archuleta, an American Idol runner-up in May, 2008. All follow a similar trajectory in user space, indicating that the users that previously listened to Girls Aloud are listening to David Archuleta a few years later.

We can also see artists like Katy Perry and Lady Gaga drift away from the center after the peak of their popularity, and we see Drake drift towards the center in what can partly be explained by a shift in his style from something more hip hop oriented to a somewhat more poppy style.

What does the variance of a trajectory indicate? The trajectories are useful not only for visualization, but also as the basis for further aggregating and quantifying the behavior of an artist. Figure 5.4 shows the artists with the small-

est and largest variance in position over time. The specific criterion used here for a given artist is the average distance over timesteps from the artist's embedding at that timestep to the mean vector of that artist's representation over all time steps. To avoid obscure artists that would be difficult to interpret without further background knowledge, we only consider artists who appeared in the track histories of at least 10% of the users.

The left-hand panel in Figure 5.4 shows the 10 artists with small variance. Many of these are well-established artists that probably undergo little change in style or fan base.

The panel in the middle and on the right-hand side of Figure 5.4 show the 10 artists with the largest variance. Many of these are popular artists that have a large change in appeal – i.e., those that go from being relatively obscure to quite popular.

The variance of a trajectory in only one possible statistic that summarizes a path. We conjecture that other summary statistics will highlight other aspects of an artist's development, providing additional criteria for exploratory data analysis.

5.3.3 User-Dynamic Model

The user-dynamic model is dual to the song-dynamic model, in that it models trajectories of users on a map of songs. While the trajectories of indiviual users provide an intersting tool for reflection, they are difficult to interpret for outsiders. We therefore only show aggregate user paths.

One such aggregation is shown in Figure 5.5. Here, we can see the behavior of users when aggregated by age. Specifically, the users are grouped by age in 2005 in order to separate the effect of a person's absolute age from the effect of the change in the average listener's taste profile.

Distinctive differences in trajectory can be seen, with the youngest group moving to north, away from Katy Perry and many other more "sugary" pop artists, and towards more dance and R&B oriented pop artists as well as the hip hop cluster which is further north, outside the figure.

The other age groups see more lateral moves and tend to be further north, even when age is fixed. The oldest age groups (where 22 to 30 and 31 to 62 were aggregated with a larger interval due to a smaller number of users in these age ranges) start very far north, and the 31 to 62 group mostly hovers around the eastern part of the figure. Outside of the figure and to the right are where many older rock bands such as the Rolling Stones and the Beatles lie, and this oldest age group is also closer to them.

5.4 Conclusion

This chapter was based largely on the work of the author and colleagues presented at ISMIR 2013 [33]. In this chapter, we adapted our embedding method approach to incorporate temporal dynamics, and we applied the resulting model to eight years of music listening history data from Last.fm in order to analyze patterns in users and songs. The visual interpretability of these models allowed us to easily pick out patterns and events in the data. Our analyses in this chapter make a strong case for the use of embeddings as visual analysis

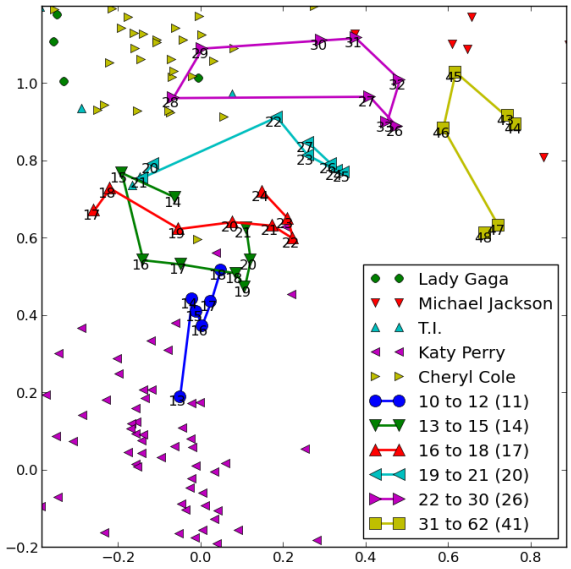

Figure 5.5: Trajectories of users with age, grouped by age in 2005. Each point is labeled with the average age of the group at that time. The legend also gives the average age in 2005 of the users in that group (in parentheses).

tools. In the next chapter, we continue this focus on data analysis, moving from temporal analysis to spatial analysis. In particular, we juxtapose our embedding taste space concept with another space – the physical geography of the earth.

CHAPTER 6

GEOGRAPHICAL, LINGUISTIC, AND CULTURAL PATTERNS IN LISTENING BEHAVIOR

In this chapter, we apply embedding methods to the gain insight about how music preferences relate to geographic, cultural, and linguistic patterns. We consider a data set of geotagged music play information extracted from tweets. These tweets are not restricted to any one country, so the data set allows us to analyze patterns in music listening behavior around the world. By modeling these tweets with embeddings, we obtain an interpretable space which gives us a way to directly measure city/city, city/artist, and artist/artist affinities. The taste space contains some surprisingly distinct segmentations across various types of borders. In particular, we find that the space of cities gives us a remarkably clear image of some cultural and linguistic phenomena that transcend geography.

6.1 Geographical Taste Space Embeddings

We focus this analysis on the relationships among cities and artists, and so we elect to condense the geographical information in a tweet down to the city from which it came. Similarly, we discard the track name from each tweet and use only the artist for the song. This leads to a joint embedding of cities and artists: the context in this case is the city from which a tweet originates and the target object is the artist which was mentioned in the tweet.

For the embedding model in this chapter, we define two embedding functions: $X(\cdot)$ for the cities and $Y(\cdot)$ for the artists. The goal then is to embed cities

and artists in a joint space such that cities are close to artists which they are likely to play. In particular, the probability for a city c to play an artist a is:

$$P(a|c) = \frac{\exp(-\Delta(X(c), Y(a)) + b_a)}{Z(c)} \tag{6.1}$$

As in previous chapters, we learn the embeddings $X(\cdot)$ and $Y(\cdot)$ by optimizing the average log-likelihood of the data given the model, and we do this using Stochastic Gradient Descent.

The resulting embedding space yields the following measures of similarity between pairs of objects:

City to Artist: this is the only similarity metric explicitly formulated in the model, and it reflects the distribution $P(a|c)$ that we directly observe data for. In particular, we directly optimize the positions of cities and artists so that cities have a high probability of listening to artists which they were observed playing in the dataset. This requires placing the city and artist nearby in the embedding space, so proximity in the embedding space can be interpreted as an affinity between a city and an artist.

Artist to Artist: due to the learned conditional probability distributions' being constrained by the metric space, two artists which are placed near each other in the space will have a similar probability mass in each city's distribution. This implies a kind of exchangeability or similarity, since any city which is likely to listen to one artist is likely to listen to the other in the model distribution.

City to City: finally, the form of similarity on which we will most rely in this work is that among cities. Again due to the metric space, two nearby cities will assign similar masses to each artist, and so will have very similar distributions

over artists in the model. This implies a similarity in musical taste or preferred artists between two cities.

The third type of similarity will form the basis for most of the analyses in this paper. In particular, we are interested in the connection between the metric space of cities in the embedding space and another metric space: the one formed by the geographic distribution of cities on the Earth's surface. As we will see, these two spaces differ greatly, and the taste space of cities gives us a clear image of some cultural and linguistic phenomena that transcend geography.

6.2 Experiments

We use the *Million Musical Tweets Dataset* (MMTD) presented by Hauger et al. [16]. This data set contains nearly 1.1 million tweets with geographical data. We pre-process the data by condensing each tweet to a city/artist pair, which results in a city/artist affinity matrix used to train the model. Next, we discard all cities and artists which have not appeared at least 100 times in the data, as well as all cities for which fewer than 30 distinct users tweeted from that city. The post-processed data contains 1,017 distinct cities and 1,499 distinct artists.

For choosing model parameters, we randomly selected 80% of the tweets for the training set, and the remaining 20% for the validation set. This resulted in a training set of 390,077 tweets and a validation set of 97,592 tweets. We used the validation set both to determine stopping criteria for the optimization as well as to choose the initial stochastic gradient step size η_0 from the set $\{0.25, 0.1, 0.05, 0.01\}$ and to evaluate the quality of models of dimension $\{2, 50, 100\}$. The optimal step size varied from model to model, but the 100-dimensional model

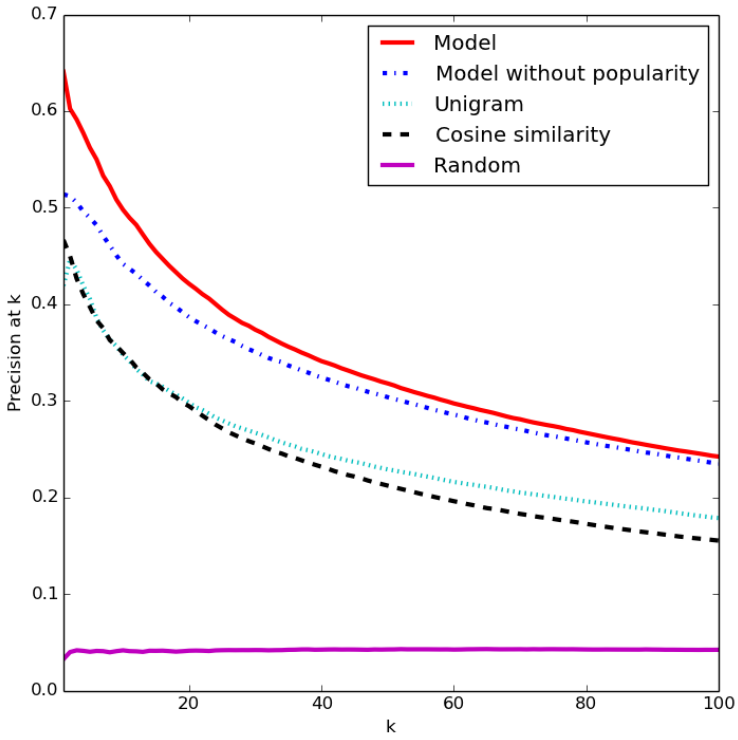

Figure 6.1: Precision at k of our model, a cosine similarity baseline, a tweet count ranking baseline, and a random baseline on a city/artist tweet prediction task.

consistently out-performed the others (although the difference between it and the 50-dimensional model was small).

We will analyze the data through the trained embedding models, both through spatial analyses (i.e. nearest neighbor queries and clusterings) and through visual inspection. In general, the high-dimensional model better captures the data, and so we will use it when direct visual inspection is not required. But first, we evaluate the quality of the model through quantitative means.

6.2.1 Quantitative Evaluation of the Model

Before we inspect our model in order to make qualitative claims about the patterns in the data, we first wish to evaluate it on a quantitative basis. This is essential in order to confirm that the model accurately captures the relations among cities and artists, which will offer validation for the conclusions we draw later in the work.

Evaluating Model Fidelity

First, we considered the performance of the model in terms of the average log-likelihood. Our baseline is the unigram distribution, which assumes that $P(a|c)$ is directly proportional to the number of tweets artist a received in the entire data set independent of the city. Estimating the unigram distribution from the training set and using it to calculate the perplexity on the validation set yielded an average log-likelihood of -6.37 (very similar to the likelihood attained when estimating this distribution from the train set and calculating the likelihood on the train set itself). Our model offered a significant improvement over this – the 100-dimensional model yielded a likelihood on the validation set of -5.67, while the 2-dimensional model reached a likelihood of -5.87. This improvement suggests that our model has captured a significant amount of useful information from the data.

Evaluating Predictive Accuracy

Second, we created a task to evaluate the predictive power of our model. To this end, we split the data chronologically into two halves, and further divided the

first half into a training set and a validation set. Using the first half of the data, we trained a 100-dimensional model. Our goal is to use this model to predict which new artists various cities will begin listening to in the second half of the data.

We accomplish this by considering, for each city, the set of artists which had no observed tweets in that city in the first half of the data. We then sorted these artists by their score in the model – namely, for city c and artist a, the function $-\Delta(X(c), Y(a)) + b_a$. Using this ordering as a ranking function, we calculated the precision at k of our ranking for various values of k, where an artist is considered to be relevant if that artist receives at least one tweet from that city in the second half of the data. We average the results of each city's ranking.

We compare the performance of our model on this task to three baselines. First, we consider a *random* ranking of all the artists which a city has not yet tweeted. Second, we sort the yet untweeted artists by their raw global tweet count in the first half of the data – which we label the *unigram* baseline. Third, we use the raw artist tweet counts for a city's nearest neighbor city in the first half of data to rank untweeted artists for that city. In this case, the nearest neighbor is not determined using our embedding but rather based on the maximum *cosine similarity* between the vector of artist tweet counts for the city and the vectors of tweet count for all other cities.

The results can be seen in Figure 6.1. At $k = 1$, our model correctly guesses an artist that a city will later tweet with 64% accuracy, compared to 46% for the cosine similarity, 42% for unigram and around 5% for the random baseline. This advantage is consistent as k increases, with our method attaining about 24% precision at 100, compared to 18% for unigram and 14% for cosine simi-

larity. We also show the performance of the same model at this task when bias terms are excluded from the scoring function at ranking time. Interestingly, the performance in this case is still quite good. We see precision at 1 of about 51% in this case, with the gap between this method and the method with bias terms growing smaller as k increases. This suggests that proximity in the space is very meaningful, which is an important validation of the analyses to follow. Finally, the good performance on this task invites an application of the space to making marketing predictions – which cities are prone to pick up on which artists in the near future? – but we leave this for future work.

6.2.2 Visual Inspection of the Embedding Space

In Figures 6.2 and 6.3 we present plots of the two-dimensional embedding space, with labels for some key cities and artists. Note that the two plots are separated by city and artists only for readability, and that all points lie in the same space. In this figure, we can already see a striking segmentation in city space, with extreme distinction between, e.g., Brazilian cities, Southeast Asian cities, and American cities. We can also already see distinct regional and cultural groupings in some ways – the U.S. cities largely form a gradient, with Chicago, Atlanta, and Philadelphia in the middle, Cleveland and Detroit on one edge of the cluster, and New York and Los Angeles on the opposite edge. This pattern is inspected in more detail in Figure 6.4, where we magnify this region of the space. In this view, we see that, compared to the larger cities in the Southeast and Midwest, many smaller cities in these regions fall even further from the Northeast and West Coast cities. However, the large cities in the Southeast and Midwest still group more closely with these smaller cities than they do with

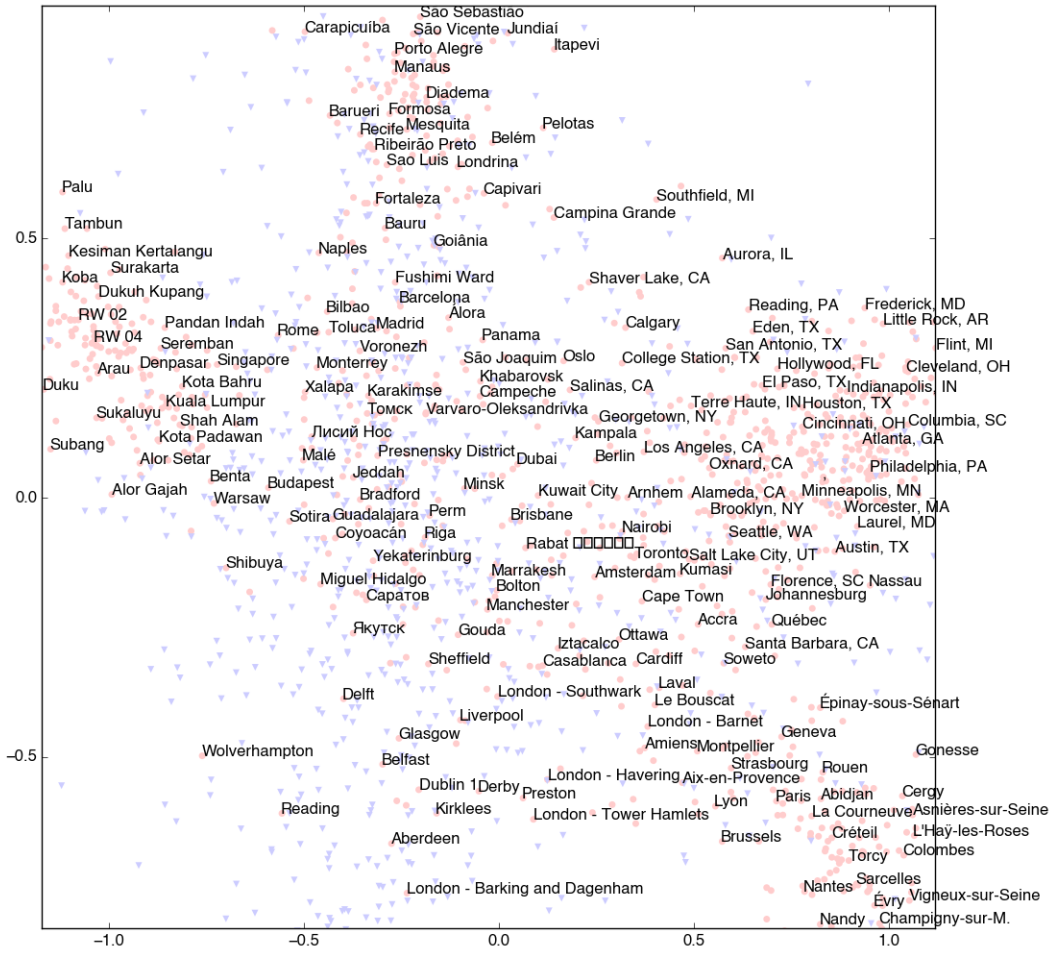

Figure 6.2: The joint city/artist space with some key cities labeled.

cities in other regions. This close-up reveals an even more thorough mixing of these two regions in the city space. Interestingly, Toronto is also on the edge of the U.S. cluster, and on the same edge where New York and Los Angeles – arguably the most "international" of the U.S. cities shown here – end up.

It is also interesting to note that the space has a very clear segmentation in terms of genre – almost as clear as the embeddings produced in previous chap-

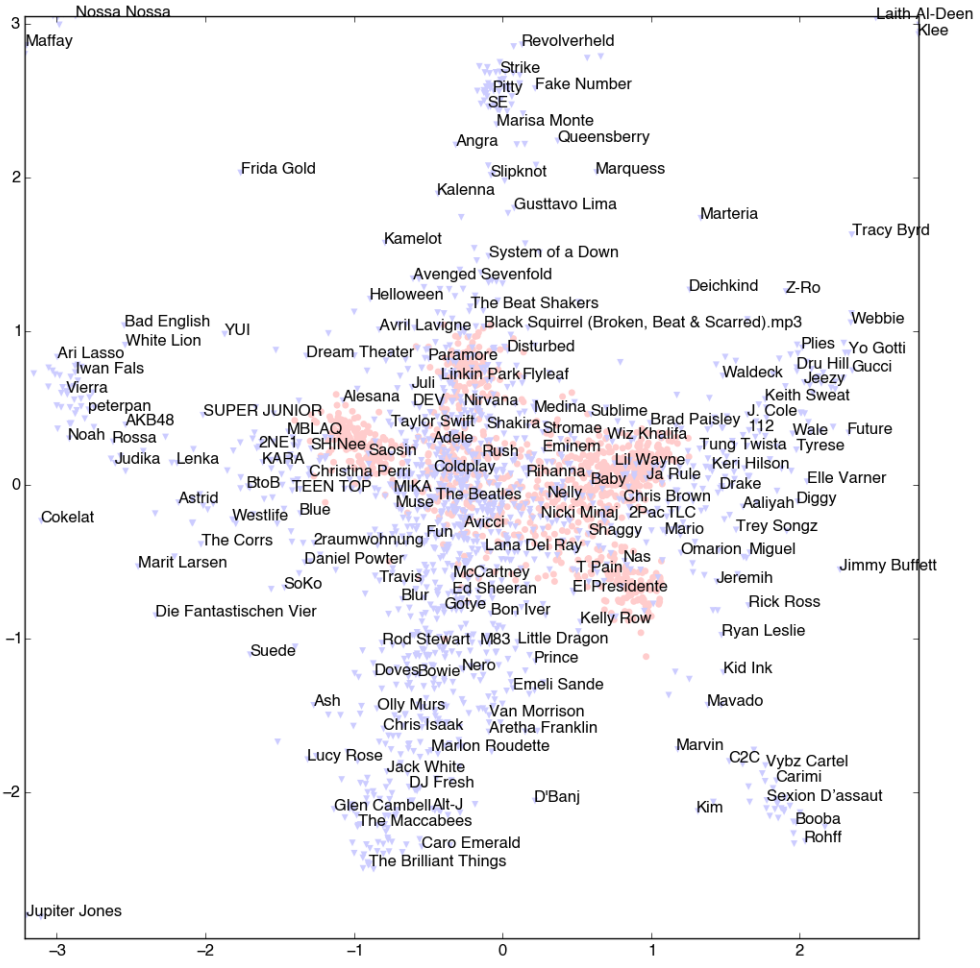

Figure 6.3: The joint city/artist space with some key artists labeled.

ters. Of course, this does not translate into an effective user model – surely there are many users in Recife, Brazil that would quickly tire of a radio station inspired by Linkin Park – but we believe it is still a meaningful phenomenon. Specifically, this suggests that the taste of the average listener can vary dramatically from one city to the next, even within the same country. More surprisingly, this variation in the average user is so dramatic that cities themselves can form nearly as coherent a taste space as individual users, as the genre segmentation

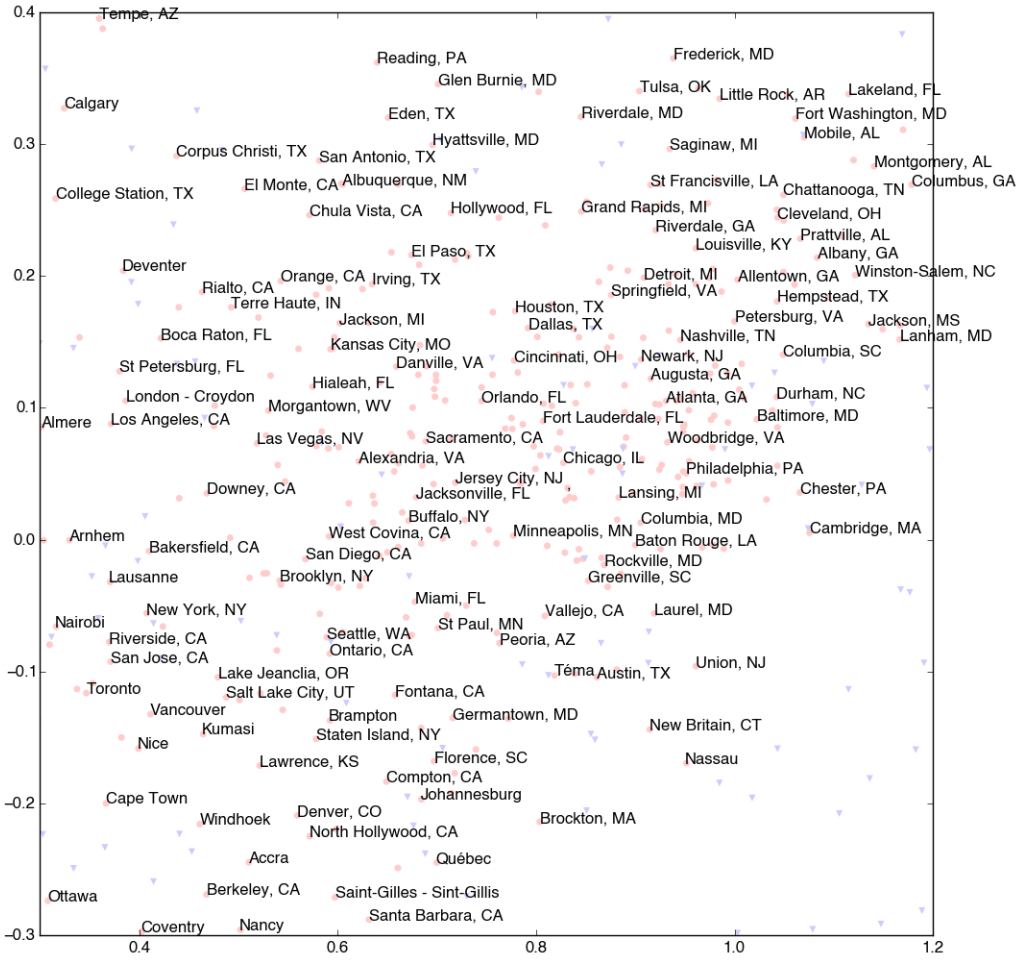

Figure 6.4: A close-up of the USA region of the city space.

is almost as clear as what is found in the models in previous chapters.

6.2.3 Higher-dimensional Models

Directly visualizing two-dimensional models can give us striking images from which rough patterns can be easily gleaned. However, higher dimensional

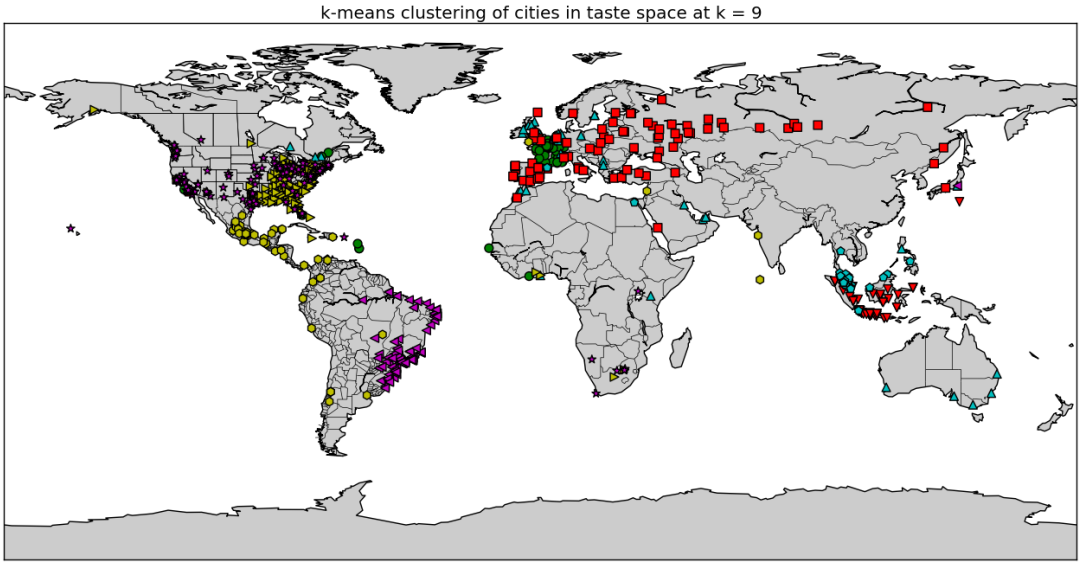

Figure 6.5: A k-means clustering of cities around the world with $k = 9$.

Kuala Lumpur	**Paris**	**Singapore**	**Los Angeles, CA**	**Chicago, IL**	**São Paulo**
Kulim	Boulogne-Billancourt	Hougang	Grand Prairie, TX	Buffalo, NY	Osasco
Sungai Lembing	Brussels	Seng Kang	Ontario, CA	Clarksville, TN	Jundiaí
Ipoh	Rennes	USJ9	Riverside, CA	Cleveland, OH	Carapicuíba
Kuching	Lille	Subang	Sacramento, CA	Durham, NC	Ribeirão Pires
Sunway City	Aix-en-Provence	Kota Bahru	Salinas, CA	Birmingham, AL	Shinjuku
Seremban	Limoges	Bangkok	Paterson, NJ	Flint, MI	Vargem Grande Paulista
Seri Kembangan	Amiens	Alam Damai	San Bernardino, CA	Montgomery, AL	Santa Maria
Taman Cheras Hartamas	Marseille	Kota Padawan	Inglewood, CA	Nashville, TN	Itapevi
Kuantan	Geneva	Glenmarie	Modesto, CA	Jackson, MS	Cascavel
Selayang	Grenoble	Budapest	Pomona, CA	Paterson, NJ	Embu das Artes
Brooklyn, NY	**Atlanta, GA**	**Madrid**	**Amsterdam**	**Sydney**	**Montréal**
Minneapolis, MN	Savannah, GA	Sevilla	Eindhoven	Toronto	Montpellier
Winston-Salem, NC	Tallahassee, FL	Granada	Tilburg	Denver, CO	Geneva
Arlington, VA	Cleveland, OH	Barcelona	Emmen	Windhoek	Raleigh, NC
Waterbury, CT	Washington, DC	Murcia	Nijmegen	Angers	Limoges
Washington, DC	Memphis, TN	Sorocaba	Enschede	Rialto, CA	Angers
Syracuse, NY	Flint, MI	Ponta Grossa	Zwolle	Hamilton	Ontario, CA
Jersey City, NJ	Huntsville, AL	Huntington Beach, CA	Amersfoort	Rotterdam	Anchorage, AK
Louisville, KY	Montgomery, AL	Istanbul	Maastricht	Ottawa	Nice
Tallahassee, FL	Jackson, MS	Vigo	Antwerp	London - Tower Hamlets	Lyon
Ontario, CA	Lafayette, LA	Oxford	Coventry	London - Southwark	Rennes

Table 6.1: Nearest neighbor query results in 100-dimensional city space. Brooklyn was chosen over New York, NY due to having more tweets in the data set. In addition, only result cities with population at least 100,000 are displayed.

models are able to achieve perplexities on the validation set which far exceed those of lower dimensional models. For example, as mentioned before, our best performing 2-dimensional model attains a validation perplexity of 357, while our best performing 100-dimensional model attains a perplexity of 290 on the validation set. This suggests that higher dimensional models capture more of the nuanced patterns present in the data. On the other hand, simple plotting is no longer sufficient to inspect high-dimensional data – we must resort to alternative methods, for example, clustering and nearest neighbor queries. First, in Figure 6.5, we present the results of using k-means clustering in the city space of the 100-dimensional model. The common algorithm for solving the k-means clustering problem is known to be prone to getting stuck in local optima, and in fact can be difficult to validate properly. We attempted to overcome these problems by using cross validation and repeated random restarts. Specifically, we used 10-fold cross-validation on the set of all cities in order to find a validation objective for each candidate value of k from 2 to 20. Then, we selected the parameter k by choosing the largest value for which no larger value offers more than a 5% improvement over the immediately previous value.

Once the value of k was chosen, we tried to overcome the problem of local optima by running the clustering algorithm 10 times on the entire set of cities with that value of k and different random initializations, finally choosing the trial with the best objective value. This process resulted in optimal k values ranging from 6 to 13. Smaller values resulted in some clusterings with granularity too coarse to see interesting patterns, while larger values were noisy and produced unstable clusterings. Ultimately, we found that $k = 9$ was a good trade-off.

Additionally, in Table 6.1, we obtain a complementary view of the 100-dimensional embedding by listing the results of nearest-neighbor queries for some well-known, hand-selected cities. These queries give us an alternative perspective of the city space, pointing out similarities that may not be apparent from the clustering alone. By combining these views, we can start to see many interesting patterns arise:

The French-speaking supercluster: French-speaking cities form an extremely tight cluster, as can also be seen in the 2-dimensional embedding in Figure 6.2. Virtually every French city is part of this cluster, as well as French-speaking cities in nearby European countries, such as Brussels and Geneva. Indeed even beyond the top 10 listed in Table 6.1, almost all of the top 100 nearest neighbors for Paris are French-speaking. Language is almost certainly the biggest factor in this effect, but if we consider the countries near France, we see that despite linguistic divides, in the clustering, many cities in the U.K. still group closely with Dutch cities and even Spanish cities. Furthermore, this grouping can be seen in every view of the data – in the two-dimensional space, the clustering, and the nearest neighbor queries. It should be noted that in our own trials clustering the data, the French cluster is one of the first clusters to become apparent, as well as one of the most consistent to appear. We can also see that the French cluster is indeed a linguistic and cultural one which is not just due to geographic proximity: although Montreal has several nearest neighbors in North America, it is present in the French group in the k-means clustering (as is Quebec City) and is also very close to many French-speaking cities in Europe, such as Geneva and Lyon. We can also see that Abidjan, Ivory Coast joins the French k-means cluster, as do Dakar in Senegal, Les Abymes in Guadeloupe and Le Lamentin and Fort-de-France in Martinique – all cities in countries which are

members of the Francophonie.

Australia: Here again, despite the relatively tight geographical proximity of Australia and Southeast Asia, and the geographic isolation of Australia from North America, Australian cities tend to group closely with Canadian cities and some cities in the United Kingdom. One way of seeing this is the fact that Sydney's nearest neighbors include Toronto, Hamilton, Ontario, Ottawa, and two of London's boroughs. In addition, other cities in Australia also belong to a cluster that mainly includes cities in the Commonwealth (e.g., U.K., Canada).

Cultural divides in the United States: the cities in the U.S. tend to form at least two distinct subgroups in terms of listening patterns. One group contains many cities in the Southeast and Midwest, as well as a few cities on the southern edge of what some might call the Northeast (Philadelphia, for example). The other group consists primarily of cities in the Northeast, on the West Coast, and in the Southwest of the country, including most of the cities in Texas. Intuitively, there are two results that might be surprising to some here. The first is that the listening patterns of Chicago tend to cluster with listening patterns in the South and the rest of the Midwest, and not those of very large cities on the coasts (after all, Chicago is the third-largest city in the country). The second is that Texas groups with the West Coast and Northeast, and not with the Southeast, which would be considered by many to be more culturally similar in many ways.

6.2.4 Most and least typical cities

We can also consider the relation of individual cities to their member countries. For this analysis, we considered all the countries which have at least 10 cities

Country	Least typical	Most typical
Brazil	Criciúma, Santa Catarina	Itapevi, São Paulo
Canada	Surrey, BC	Toronto, ON
Netherlands	Leiden	Emmen
Mexico	Campeche, CM	Cuauhtémoc, DF
Indonesia	Panunggangan Barat	RW 02
France	Bordeaux	Mantes-la-Jolie, Île-de-France
United States	Huntington Beach, CA	Jackson, MS
Malaysia	Kota Damansara	Kuala Lumpur
United Kingdom	Wolverhampton, England	London Borough of Camden
Russia	Ufa	Podgory
Spain	Álora, Andalusia	Barcelona

Table 6.2: Most and least typical cities in taste profile for various countries.

represented in the data. Then for each country we calculated the average position in embedding space of cities in that country. With this average city position, we can then measure the distance of individual cities from the mean of cities in their country and find the cities which have the most and least typical taste profiles for that country.

The results are shown in Table 6.2. We can see a few interesting patterns here. First, in Brazil, the most typical city is an outlying city near São Paulo city, while the least typical is a city in Santa Catarina, the second southernmost state in Brazil, which is also less populous than the southernmost, Rio Grande do Sul, which was also well-represented in the data. In Canada, the least typical city is an edge city on Vancouver's east side, while the most typical is the largest city, Toronto. In France, the most typical city is in Île-de-France, not too far from Paris. We also see in England that the least typical city is Wolverhampton, an edge city of Birmingham towards England's industrial north, while the most typical is a borough of London.

6.3 Conclusions

In this chapter, we expanded our embedding-based data analysis to geographical data. The work presented here was presented in prior work by the author and colleagues at ISMIR 2014 [34]. Using embeddings, we were able to easily process a large amount of data and sift through it visually and with spatial analysis in order to uncover examples of how musical taste conforms to or transcends geography, language, and culture. Our findings reflect that differences in culture and language, as well as historical affinities among countries otherwise separated by vast distances, can be seen very clearly in the differences in taste among average listeners from one region to the next. More generally, this analysis further shows how nuanced patterns in large collections of preference data can be condensed into a taste space, which provides a powerful tool for discovering complex relationships in unstructured data.

CHAPTER 7

CONCLUSION

Embedding methods offer a way to model data in order to obtain a visually interpretable semantic spaces. These can be used for generative modeling tasks as well as data analysis. As has been shown in this work, these methods are modular as well as meaningful. It is easy to adapt embeddings to many different task domains and modeling problems, and in each case the model produces a space which can easily be visualized in order to understand the model.

In particular, we have discussed the domain of music information retrieval at length. This is a pertinent topic for making sense of log data, since music streaming services continue to grow in popularity and continue to expand their already massive music libraries. In addition to demonstrating the general power of embedding methods for modeling tasks, the contributions in this work include substational contributions to the field of MIR. First, we demonstrated the applicability of such methods to the task of playlist generation. We also showed how to extend our models to incorporate side information, enabling us to overcome the Out-of-Vocabulary (OOV) problem, where songs that were not seen in training cannot be accomodated at test time.

We then showed how our methods can enable analysis of music listening behavior data sets. First, we extended our model to include temporal dynamics in order to analyze a long-term data set of users' track play logs. With these methods, we were able to easily and intuitively detect important trends and events in the data. Second, we applied our methodology to the case of a data set of music listening behavior in cities around the world. By embedding cities and artists in a joint space and taking advantage of the similarity metrics yielded

by this space, we were able to find rich cultural, geographical, and linguistic patterns in the data.

In each of these cases, embeddings offered a modular, easily adaptable, semantically rich model for understanding data. These models could have interesting applications in a number of domains, including language, social networks, purchase histories, and more. The specific models and extensions developed in this work offer a solid foundation and a number of ideas which could be utilized by future works in these domains.

BIBLIOGRAPHY

[1] N. Aizenberg, Y. Koren, and O. Somekh. Build your own music recommender by modeling internet radio streams. In *WWW*, pages 1–10. ACM, 2012.

[2] L. Barrington, R. Oda, and G. Lanckriet. Smarter than genius? human evaluation of music recommender systems. *ISMIR*, 2009.

[3] Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. Neural probabilistic language models. *Innovations in Machine Learning*, pages 137–186, 2006.

[4] Yoshua Bengio and J-S Senecal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *Neural Networks, IEEE Transactions on*, 19(4):713–722, 2008.

[5] Yoshua Bengio, Jean-Sébastien Senécal, et al. Quick training of probabilistic neural nets by importance sampling. In *AISTATS Conference*, 2003.

[6] S. Chen, J. L. Moore, D. Turnbull, and T. Joachims. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 714–722. ACM, 2012.

[7] Shuo Chen, Jiexun Xu, and Thorsten Joachims. Multi-space probabilistic sequence modeling. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 865–873. ACM, 2013.

[8] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

[9] Trevor F Cox and Michael AA Cox. *Multidimensional scaling*. CRC Press, 2000.

[10] Mathias Creutz, Teemu Hirsimäki, Mikko Kurimo, Antti Puurula, Janne Pylkkönen, Vesa Siivola, Matti Varjokallio, Ebru Arisoy, Murat Saraçlar, and Andreas Stolcke. Morph-based speech recognition and modeling of out-of-vocabulary words across languages. *ACM Transactions on Speech and Language Processing (TSLP)*, 5(1):3, 2007.

[11] G. Dror, N. Koenigstein, and Y. Koren. Yahoo! music recommendations: modeling music ratings with temporal dynamics and item taxonomy. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 165–172. ACM, 2011.

[12] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2010.

[13] David F. Gleich, Leonid Zhukov, Matthew Rasmussen, and Kevin Lang. The World of Music: SDP embedding of high dimensional data. In *Information Visualization 2005*, 2005.

[14] A. Globerson, G. Chechik, F. Pereira, et al. Euclidean embedding of co-occurrence data. *Journal of Machine Learning Research*, 2007.

[15] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[16] D. Hauger, M. Schedl, A. Košir, and M. Tkalčič. The million musical tweets dataset - what we can learn from microblogs. In *ISMIR*, 2013.

[17] David Hauger and Markus Schedl. Exploring geospatial music listening patterns in microblog data. In *Adaptive Multimedia Retrieval: Semantics, Context, and Adaptation*, pages 133–146. Springer, 2014.

[18] V. Jain and L. Saul. Exploratory analysis and visualization of speech and music by locally linear embedding. *ICASSP*, 2004.

[19] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[20] Sanghoon Jun, Seungmin Rho, and Eenjun Hwang. Geographical region mapping scheme based on musical preferences. In *ISMIR*, 2014.

[21] P. Knees, T. Pohle, M. Schedl, and G. Widmer. Combining audio-based similarity with web-based data to accelerate automatic music playlist generation. *ACM MIR*, 2006.

[22] I. Knopke. Geospatial location of music and sound files for music information retrieval. *ISMIR*, 2005.

[23] B. Logan. Content-based playlist generation: exploratory experiments. *ISMIR*, 2002.

[24] F. Maillet, D. Eck, G. Desjardins, and P. Lamere. Steerable playlist generation by learning song similarity from radio station playlists. In *International Conference on Music Information Retrieval (ISMIR)*, 2009.

[25] Y. Maron, M. Lamar, and E. Bienenstock. Sphere embedding: An application to part-of-speech induction. In *Neural Information Processing Systems Conference (NIPS)*, 2010.

[26] Brian McFee and Gert R. G. Lanckriet. The natural language of playlists. In *International Conference on Music Information Retrieval (ISMIR)*, 2011.

[27] Brian McFee and Gert RG Lanckriet. Hypergraph models of playlist dialects. In *ISMIR*, pages 343–348. Citeseer, 2012.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[29] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[30] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems*, pages 2265–2273, 2013.

[31] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *ICML*, 2012.

[32] J. L. Moore, S. Chen, T. Joachims, and D. Turnbull. Learning to embed songs and tags for playlist prediction. In *ISMIR*, 2012.

[33] J. L. Moore, Shuo Chen, T. Joachims, and D. Turnbull. Taste over time: the temporal dynamics of user preferences. In *ISMIR*, 2013.

[34] J. L. Moore, T. Joachims, and D. Turnbull. Taste space versus the world: an embedding analysis of listening habits and geography. In *ISMIR*, 2014.

[35] J. Platt. Fast embedding of sparse music similarity graphs. *NIPS*, 2004.

[36] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[37] Lawrence K Saul and Sam T Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *The Journal of Machine Learning Research*, 4:119–155, 2003.

[38] Markus Schedl and David Hauger. Mining microblogs to infer music artist similarity and cultural listening patterns. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 877–886. ACM, 2012.

[39] U. Shalit, D. Weinshall, and G. Chechik. Modeling musical influence with topic models. In *ICML*, 2013.

[40] J. Weston, S. Bengio, and P. Hamel. Multi-tasking with joint semantic spaces for large-scale music annotation and retrieval. *JNMR*, 40(4):337–348, 2011.

[41] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2764–2770. AAAI Press, 2011.